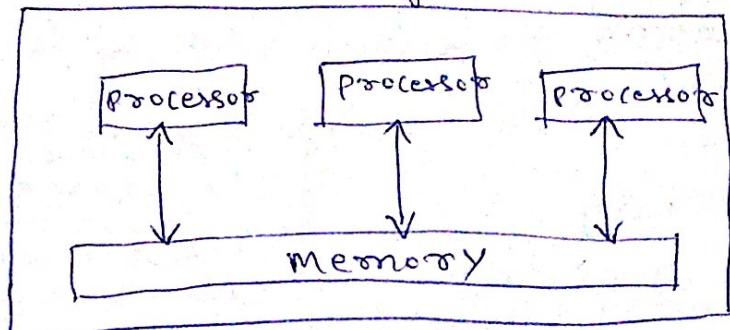


parallel and distributed computing

parallel computing

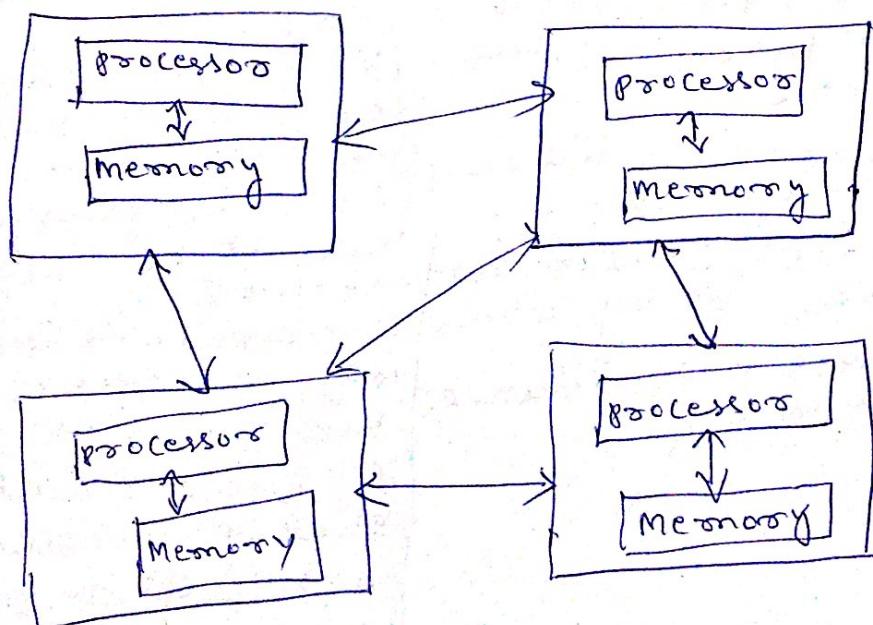
A parallel computing system consists of multiple processors that communicate with each other using a shared memory.



parallel computing

Distributed Computing

A distributed computing system contains multiple processors connected by a communication network.



In distributed computing system, multiple system processors can communicate with each other using messages that are send over the network.

- Such systems are increasingly available these days because of the availability at low price of computer processors and high-bandwidth links to connect them.

Examples of distributed systems / applications of distributed computing

intranets, internet, www, email

Telecommunication networks - Telephone networks and cellular networks.
Organizations such as Facebook and Google widely use distributed computing to allow users to share resources.

Parallel computing

Type of computation in which many calculations or the execution of process are carried out simultaneously

i - occurs in a single computer

ii - multiple processors execute tasks at the same time

iii - computer can have shared memory

iv - processes communicate each other using a bus

v - increase performance of the system

VS

Distributed computing

A system whose components are located on different networked computers which communicate and coordinate their action by passing messages to one another

ii - involves multiple computers

iii - multiple computers perform tasks at the same time

iv - each computer has its own local memory.

v - computers communicate with each other via the network

vi - allow scalability, sharing resources, and helps to perform computation tasks efficiently.

* The following reasons explain why a system should be built distributed not just parallel

→ Scalability - All distributed systems do not have the problem associated with the shared memory, with the increased number of processors

obviously regarded as they are more scalable than parallel systems.

- Reliability — the impact of failure of any single subsystem or a computer on the network of computers defines the reliability of such a connected system so distributed system is better reliable than parallel system.
- Data sharing — data sharing provided by distributed systems is similar to data sharing provided by distributed database.
- Resource sharing — if there exist an expensive and a special purpose resource or a processor, which can't be dedicated to each processor in the system, such a resource can be easily shared across distributed system.
- Heterogeneity — A system should be flexible enough to accept a new heterogeneous processor to be added into it and one of the processors to be replaced or removed from the system without affecting the overall system processing capabilities.
- The geographic placement of different subsystems of an application may be inherently placed as distributed.
- Economic — with the evolution of modern computers, high-bandwidth networks and workstations are available at low cost, which also favors distributed computing for economic reasons.

* number of computers — parallel computing occurs in a single computer whereas distributed computing involves multiple computers.

Functionality — In parallel computing multiple processors execute multiple tasks at the same time. However in distributed computing, multiple computers perform tasks at the same time.

Memory — In parallel computing the computer can have a shared memory; in distributed computing each computer has its own memory.

Usage — Parallel computing helps to increase the performance of the system. Distributed computing allows scalability, sharing resources and helps to perform computation tasks efficiently.

Parallel Computing

Parallel computing is also called parallel processing.

- There are multiple processors in parallel computing.
- Each of them performs the computations assigned to them.
- In other words, in parallel computing multiple calculations are performed simultaneously.

Advantages

- There are multiple advantages to parallel computing.

As there are multiple processors working simultaneously, it increases the CPU utilization and improves the performance.

- Failure of one processor does not affect the functionality of other processors. Therefore parallel computing provides reliability.

parallel computing and distributed computing are two types of computation

* the main difference between parallel computing and distributed computing is that parallel computing allows multiple processors to execute tasks simultaneously.

While distributed computing divides a single task between multiple computers to achieve a common goal.

Need for parallel computing

It is the use of multiple processing elements simultaneously for solving any problem.

- problems are broken down into instructions and are solved concurrently as each resource which has been assigned to work is working at the same time.

Advantages -

- it saves time and money as many resources working together at once will reduce the time and cut potential costs.
- it can be impractical to solve larger problems on serial computing.

Why parallel computing

- The whole real world runs in dynamic nature that is many things happens at a certain time but at different places concurrently. This data is extensively huge to manage.

- Realworld data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is key.

- parallel computing provides concurrency and saves time and money.
- complex, large datasets and their management can be organized only and only using parallel computing approach.
- Ensures the effective utilization of the resources. The hardware is used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.
- it is impractical to implement real-time systems using serial computing.

Applications of parallel computing

- data base and data-mining.
- Real time simulation of systems
- science and engineering.
- Advance graphics, augmented reality and virtual reality.

$$y = (u \times 5) + (1 \times 6) + (5 \times 3)$$

on a single processor, the steps needed to calculate a value for y might look like:

$$\text{Step 1: } y = 20 + (1 \times 6) + (5 \times 3)$$

$$\text{Step 2: } y = 20 + 6 + (5 \times 3)$$

$$\text{Step 3: } y = 20 + 6 + 15$$

$$\text{Step 4: } y = 41$$

but in parallel computing scenario with three processors or computers

the steps look something like:

$$\rightarrow \text{step 1: } y = 20 + 6 + 15$$

$$\rightarrow \text{step 2: } y = 41$$

break the task down into pieces and execute those pieces simultaneously.

MODELS OF COMPUTATION

concurrent processing

it can divide a complex task and process it multiple systems to produce the output in quick time.

concurrent processing is essential where the task involves processing a huge bulk of complex data.

Ex: accessing large databases, aircraft testing, astronomical calculations, atomic and nuclear physics, biomedical analysis, economic planning, image processing, robotics, weather forecasting, Web based services etc.

What is parallelism

it is a process of processing several set of instructions simultaneously.

→ it reduces the total computational time

→ parallelism can be achieved by using of parallel computers. That is a computer with many processors.

→ parallel computer requires parallel algorithms, programming languages, compilers and operating systems that supports multitasking.

* Algorithm is a sequence of instructions followed to solve a problem. So while designing an algorithm, we should consider the architecture of computer on which algorithm will be executed.

- as per the architecture there are two types of computers -
- Sequential Computer
 - Parallel Computer.

Depending on the architecture of computers, we have two types of algorithm.

→ Sequential Algorithm - An algorithm in which some consecutive steps of instructions are executed in a chronological orders to solve a problem.

→ parallel algorithms - The problem is divided into sub-problems and are executed in parallel to get individual output. Later on, these individual outputs are combined together to get the final desired output.

To design an algorithm properly, we must have a clear idea of the basic model of computation in a parallel computer.

Models of Computation

both Sequential and parallel computers operate on a set of instructions called algorithm.

→ these set of instructions (algorithm) instruct the computer about what it has to do in each step.

Depending on the instruction stream and data stream, computers can be classified into four categories.

I → Single Instruction Stream, Single Data Stream (SISD) computers

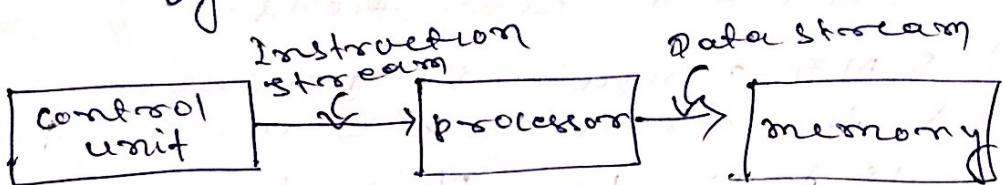
II → Single Instruction Stream, Multiple Data Stream (SIMD) computers

III → multiple Instruction Stream, multiple single data Stream (misd) computers

IV → multiple Instruction Stream, multiple data stream (mim) computers.

SISD computers —

SISD computers contain one control unit, one processing unit and one memory unit.

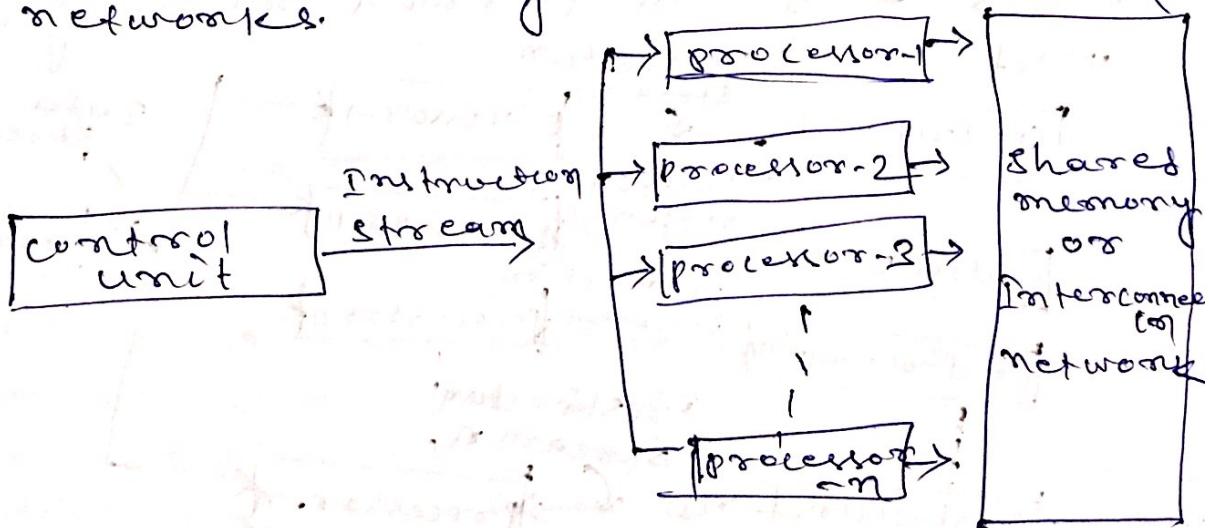


In this type of computers, the processor receives a single instruction stream from the control unit and operates on single stream of data from the memory unit.

→ During computation, at each step, the processor receives one instruction from the control unit and operates on a single data received from the memory unit.

SIMD computers —

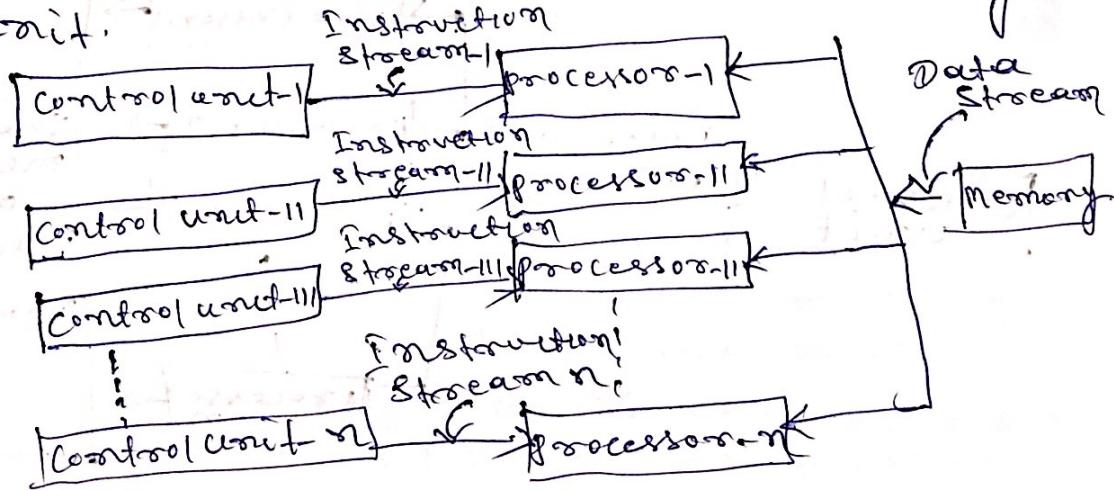
SIMD computers contain one control unit, multiple processing units, and shared memory or interconnection networks.



- here one single control unit ~~sends~~ sends instructions to all processing units.
- During computation, at each step, all processors receive a single set of instructions from the control unit and operates on different set of data from the memory unit.
- each of the processing units has its own local memory unit to store both data and instructions.
- In SIMD computers, processors need to communicate among them. This is done by shared memory or by interconnection networks.
- While some of the processors execute a set of instructions, the remaining processors wait for their next set of instructions. Instruction flow control unit decides which processor will be active (execute instructions) or inactive (wait for the next instruction).

MISD Computers

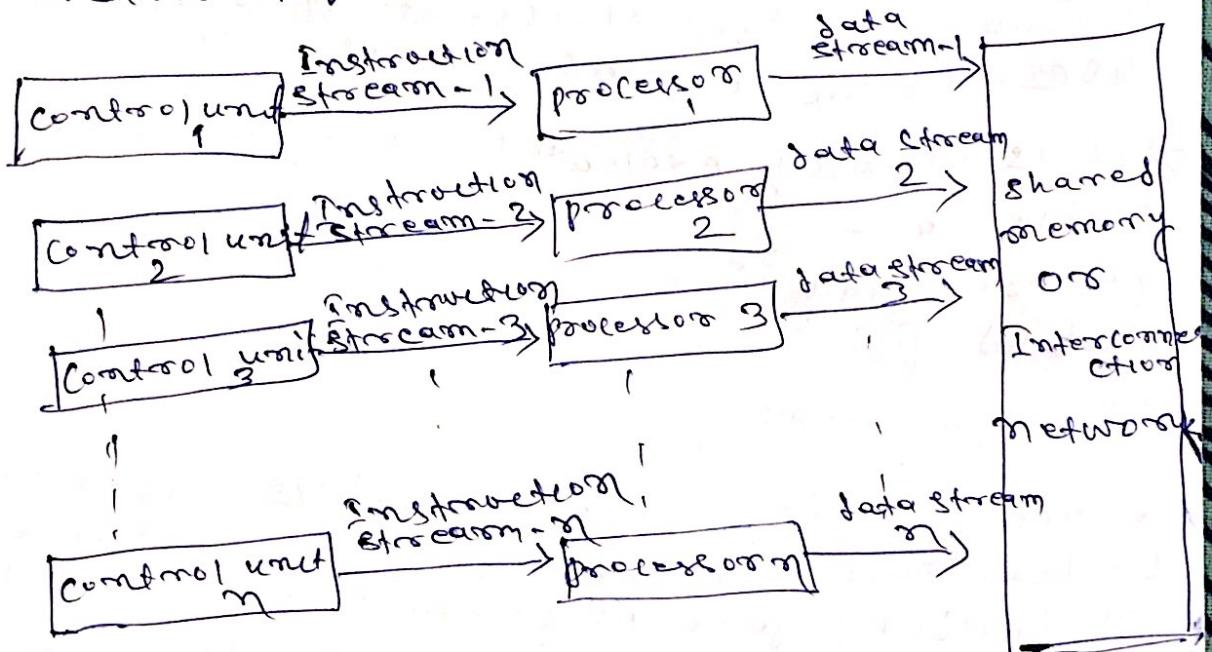
MISD computers contain multiple control units, multiple processing units and one common memory unit.



→ here each processor has its own control unit and they share a common memory unit. All the processors get instructions individually from their own control units and they operate on a single stream of data as per the instructions they have received from their respective control units. This processor operates simultaneously.

MIMD Computers

MIMD computers have multiple control units, multiple processing units and a shared memory or interconnection network.



Here each processor has its own control unit, local memory unit and arithmetic and logic unit.

→ They receive different sets of instructions from their respective control units and operate on different sets of data.

* An MIMD computer that shares a common memory is known as multiprocessor while those that use an interconnection network is known as multi-computer.

Multi-computer - when all the processors are very close to one another (in the same room)

Distributed system - when all the processors are far away from one another (in different cities)

ANALYZING PARALLEL ALGORITHMS

→ This refers to the process of determining how good an algorithm is, that is how fast, how expensive to run, and how efficient it is in its use of available resources.

→ It is usually evaluated using the following criteria:-

- (i) running time (ii) number of processors used (iii) cost

Running time

Since speeding of computations appears to be the main reason behind our interest in building parallel computers, the most important measure is evaluating a parallel algorithm is therefore its running time.

→ It is defined as the time taken by the algorithm to solve a problem on a parallel computer. That is the time elapsed from the moment the algorithm starts to the moment it terminates. If the various processors do not all begin and end their computations

simultaneously, then the running time between the moment is equal to the time elapsed, the moment is equal to the time elapsed for the moment the first processor to begin computing starts and the moment the last processor to end computing terminates.

* Counting steps - before actually implementing an algorithm (whether sequential or parallel) on a computer, it is necessary to conduct a theoretical analysis of the time it will require to solve the computational problem to solve the problem as executed by the algorithm at worst case.

The running time of a parallel algorithm usually obtained by counting two kinds of steps

- (i) Computational steps
- (ii) Routing steps

Computational steps — a computational step is an arithmetic or logic operation performed on a piece of information with in a processor.

* Routing steps —

the piece of information travels from one processor to another via shared memory or through the communication network.

PX we want to compute the product of two $n \times n$ matrices. Since the resulting matrix has n^2 entries, at least this many steps are needed by any matrix multiplication algorithm simply to produce the output.

Speedup

is evaluating a parallel algorithm for a given problem.

Speedup = Worst case running time of fastest known sequential algorithm for problem

worst case running time of parallel algorithm

clearly larger the speed-up, the better the parallel algorithm.

Number of Processors

The second most important criteria in evaluating a parallel algorithm is the number of processors it requires to solve a problem.

→ it costs money to purchase, maintain and run the computer. When several processors are present the problem of maintenance, in particular, is compounded and the price paid to guarantee a high degree of reliability rises sharply.

Therefore the larger the number of processors an algorithm uses to solve a problem, the more expensive the solution becomes to obtain.

Def For problem of size n , the number of processors required by an algorithm a function of n , will be denoted by $p(n)$.

Sometimes the number of processors is a constant independent

Cost

The cost of a parallel algorithm is defined as the product of the previous two measures

$$\text{cost} = \text{parallel running time} \times \text{number of processes used.}$$

$$\text{Efficiency} = \frac{\text{parallel running time}}{\text{of fastest known sequential algorithm for problem}}$$

Cost of parallel algorithm.

usually efficiency ≤ 1 otherwise a sequential algorithm can be faster than the parallel one, obtained from the parallel one.

Expressing parallel Algorithms

A parallel algorithm will normally consist of two kinds of operations: sequential and parallel.

→ if we consider the algorithm is written in structured programming language such as Pascal.

Examples of statements include if...then...else, while...do, for...do, assignment statements, inputs and outputs statements and so on.

The meaning of these statements assumes to be known.

$a \leftarrow b$ means that the value of b is assigned to a

The logical operations and, or, xor
and not are used in their familiar
connotation.

Thus if a and b are two expression
each taking one of the values true or
false, then

- (i) $(a \text{ and } b)$ is true if both a and b are
true; otherwise $(a \text{ and } b)$ is false.
- (ii) $(a \text{ or } b)$ is true if at least one of
 a and b is true; otherwise $(a \text{ or } b)$ is false.
- (iii) $(a \neq b)$ is true if exactly one
of a and b is true; otherwise
 $(a \neq b)$ is false; and
- (iv) $(\text{not } a)$ is true if a is false;
otherwise $(\text{not } a)$ is true;

parallel operations on the other hand
are expressed by two kind of statements

(i) When several steps are to be done
at the same time, we write

do steps i to j in parallel

step i

step i+1

step j. \square

(ii) When several processes are to
perform the same operation simultaneously
we write

for i=j to k do in parallel

{ The operations to be performed by P_i
are stated here }

end for \square

where i taken, every integer value from j to k, or

for i = s, ..., t do in parallel

{ operations to be performed

by P_i stated here}

end for □

where the integer values taken by i are

enumerated, or

for all i in S do in parallel

{ operations to be performed by P_i

are stated here}

end for □

Where S is a given set of integers

components ~~are~~ with curly brackets are surrounded with curly brackets

{} ,

MATRIX-MATRIX MULTIPLICATION

USING PARALLEL PROCESSORS

Let each processing elements P_{Eij} represent two elements a_{ij}, b_{ij} .

- There are only n processing elements containing a pair of scalars suitable for multiplication.

- Devide matrices A and B so that every processor has a pair of scalars that needed to be multiplied.

- The element A will move in leftward rotation and the elements of B moved in upward rotation.

- These movements present each pair with a new pair of values to be multiplied

Step-1

divide the matrix onto different parts

- The second computes all products $a_{ik} \times b_{kj}$ and accumulates sum when the phase II is completed.

Algorithm

procedure MATRIXMULT

begin

for $k=1$ to $n-1$ step 1 do

begin

for all P_{ij} , where i and j ranges from 1 to n do

if i is greater than k then

rotate a_{ij} in the east direction

end if

if j is greater than k then

rotate b in south direction

end if

end

for all P_{ij} where i and j lies before k to n do

compute the product of a and b are stored in C

for $k=1$ to $n-1$ step 1 do for all

P_{ij} where i and j ranges from 1 to $n-k$

rotate a in the east

rotate b in the south

$$C = C + a \times b$$

end

$$\underline{\text{Ex}} \quad A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

The matrix elements are stored in two dimensional mesh

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

mesh = $\begin{pmatrix} 1/5 & 2/6 \\ 3/7 & 4/8 \end{pmatrix}$

Let the number of processor be 4

$$\text{such that } n^2 = 4 \Rightarrow n = 2$$

No of processors be 4. Let

$$P_{(1,1)}, P_{(1,2)}, P_{(2,1)}, P_{(2,2)} \text{ let } k$$

ranges from 1 to n-1

According to algorithm $P_{(1,2)}$, $P_{(2,1)}$ and

$P_{(2,2)}$ will carry out the movements

as according to algorithm if i_2 is greater than k and rotate a_{21} the left (east) direction

$\rightarrow a_{21}$ and a_{22} changes its place from leftwards (row wise)

\rightarrow and b_{12} and b_{22} also changes its place from bottom to top (South direction)

mesh = $\begin{pmatrix} 1/5 & 2/6 \\ 3/7 & 4/8 \end{pmatrix} = \begin{pmatrix} 1/5 & 2/8 \\ 4/7 & 4/6 \end{pmatrix}$

leftward (row wise)

The processes P_{ij} ranges from 1 to 2 will compute the product of a_{ij} and b_{ij} respectively

$$C_{ij} = \begin{pmatrix} 1 \times 5 & 2 \times 8 \\ 4 \times 7 & 3 \times 6 \end{pmatrix} = \begin{pmatrix} 5 & 16 \\ 28 & 18 \end{pmatrix}$$

Next all P_{ij} will be move its a_{ij} and b_{ij} left and up respectively

$$\text{mesh} = \begin{pmatrix} 1/5 & 2/8 \\ 4/7 & 3/6 \end{pmatrix}$$

up shift (column wise)

mesh left shift (row wise)

$$\text{mesh}^S = \begin{pmatrix} 2/7 & 1/6 \\ 3/5 & 4/8 \end{pmatrix} = \begin{pmatrix} 14 & 6 \\ 15 & 32 \end{pmatrix}$$

$$C_{ij} = \begin{pmatrix} 14 & 6 \\ 15 & 32 \end{pmatrix}$$

Now the processes P_{ij} compute the product of a_{ij} and b_{ij} respectively with old value of C_{ij}

$$C_{ij} = \begin{pmatrix} 5 & 16 \\ 28 & 18 \end{pmatrix} + \begin{pmatrix} 14 & 6 \\ 15 & 32 \end{pmatrix}$$

$$(C_{ij})^R = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} = \text{Resultant matrix}$$

The resultant product matrix is $\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$

Ex-2

$$A = \begin{bmatrix} 2 & 1 & 5 & 3 \\ 6 & 7 & 1 & 6 \\ 9 & 2 & 4 & 4 \\ 3 & 6 & 7 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 4 & 5 & 6 & 5 \\ 1 & 9 & 8 & -8 \\ 4 & 0 & -8 & 5 \end{bmatrix}$$

If we consider a sequential algorithm the computation will perform sequentially element by element and stores the resultant in memory and then add individual elements.

$$\left[2 \times 6 + 1 \times 4 + 5 \times 1 + 3 \times 4 \right]$$

Perform element by element multiplication individually 2×6 and store the resultant value in memory and $1 \times 4, 5 \times 1, 3 \times 4$ will be done individually multiplication will be done in memory and store the value in different parts and then by adding the different parts to form a single one.

$$= 12 + 4 + 5 + 12 = 33$$

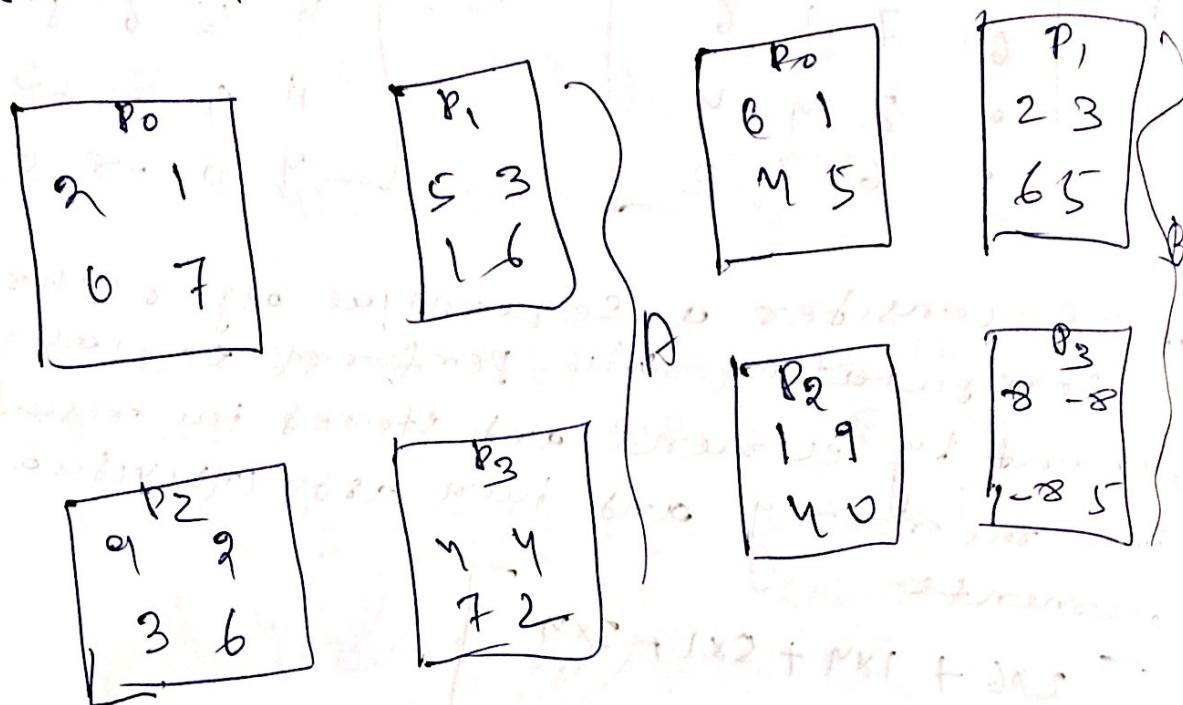
but if we consider in matrix-matrix multiplication in parallel, processes

step 1

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 2 & 1 & 5 & 3 \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 6 & 7 & 1 & 6 \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 4 & 4 & 4 & 4 \\ a_{41} & a_{42} & a_{43} & a_{44} \\ 3 & 6 & 7 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ 6 & 1 & 2 & 3 \\ b_{21} & b_{22} & b_{23} & b_{24} \\ 4 & 5 & 6 & 5 \\ b_{31} & b_{32} & b_{33} & b_{34} \\ 1 & 9 & 8 & -8 \\ b_{41} & b_{42} & b_{43} & b_{44} \\ 4 & 0 & -8 & 5 \end{bmatrix}$$

assign processes to see both the
divided matrix A and B
Suppose there are 4 processes

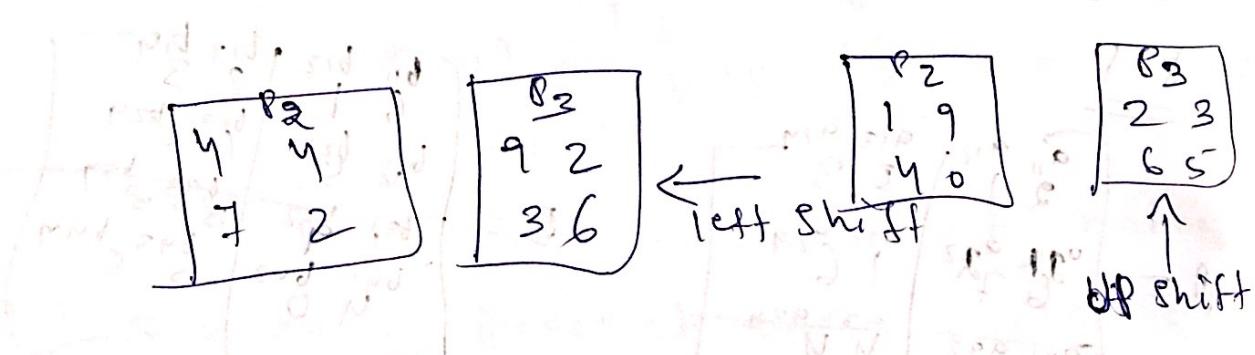
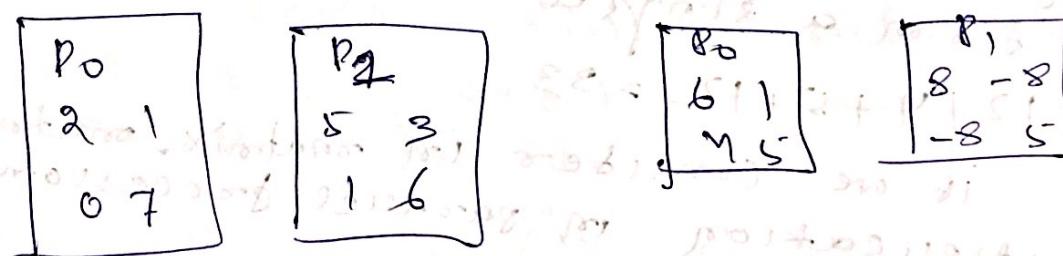


② suppose there are 4 processes

$$n^2=4 \Rightarrow n=\sqrt{4}=2 \text{ steps are}$$

required for computation

Apply left shift and VP shift for the
matrix A and B (row wise)
left shift (row wise)
VP shift (column wise)



Step - I

Apply computation \rightarrow multiplying process on
P₀ of matrix A to P₀, P₁ of matrix B

resultant matrix

$$C_0 = \begin{bmatrix} P_0 \\ 2 & 1 \\ 0 & 7 \end{bmatrix} \times \begin{bmatrix} P_0 \\ 6 & 1 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 16 & 7 \\ 28 & 55 \end{bmatrix}$$

Similarly C₁ =

$$\begin{bmatrix} P_1 \\ 5 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} P_1 \\ 8 & -8 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 16 & -24 \\ -40 & 22 \end{bmatrix}$$

$$C_2 = \begin{bmatrix} P_2 \\ 4 & 4 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} P_2 \\ 1 & 9 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix}$$

$$C_3 = \begin{bmatrix} P_3 \\ 9 & 2 \\ 3 & 6 \end{bmatrix} \times \begin{bmatrix} P_3 \\ 2 & 3 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 30 & 37 \\ -42 & 39 \end{bmatrix}$$

Step - II

Again apply left shift and upshift to all the
processes of matrix A and B

$$\begin{bmatrix} P_0 \\ 5 & 3 \\ 1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} P_1 \\ 2 & 1 \\ 0 & 7 \end{bmatrix}$$

$$\begin{bmatrix} P_0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} P_1 \\ 2 & 3 \\ 6 & 5 \end{bmatrix}$$

$$\begin{bmatrix} P_2 \\ 4 & 4 \\ 7 & 2 \end{bmatrix}$$

$$\begin{bmatrix} P_2 \\ 4 & 4 \\ 7 & 2 \end{bmatrix}$$

left shift

$$\begin{bmatrix} P_2 \\ 6 & 1 \\ 4 & 5 \end{bmatrix}$$

$$\begin{bmatrix} P_3 \\ 8 & -8 \\ -8 & 5 \end{bmatrix}$$

upshift upshift

Matrix Computation

$$C_0 = \begin{bmatrix} 80 \\ 53 \\ 16 \end{bmatrix} \times \begin{bmatrix} 19 \\ 40 \end{bmatrix} = \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} P_1 \\ 21 \\ 07 \end{bmatrix} \times \begin{bmatrix} 81 \\ 23 \\ 65 \end{bmatrix} = \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix}$$

$$C_2 = \begin{bmatrix} P_2 \\ 92 \\ 36 \end{bmatrix} \times \begin{bmatrix} P_2 \\ 61 \\ 45 \end{bmatrix} = \begin{bmatrix} 62 & 19 \\ 42 & 20 \end{bmatrix}$$

$$C_3 = \begin{bmatrix} P_3 \\ 44 \\ 72 \end{bmatrix} \times \begin{bmatrix} P_3 \\ 8 -8 \\ 25 \end{bmatrix} = \begin{bmatrix} 0 & -12 \\ 40 & -46 \end{bmatrix}$$

Resultant matrix

= Adding of step-II of C_0 with C_0 of step-III
and all C_1, C_2, C_3 are calculated in this manner.

$$C_0 = \begin{bmatrix} 16 & 7 \\ 28 & 55 \end{bmatrix} + \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix} = \begin{bmatrix} 33 & 52 \\ 53 & 64 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 16 & -25 \\ 40 & 22 \end{bmatrix} + \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix} = \begin{bmatrix} 26 & -14 \\ 42 & 57 \end{bmatrix}$$

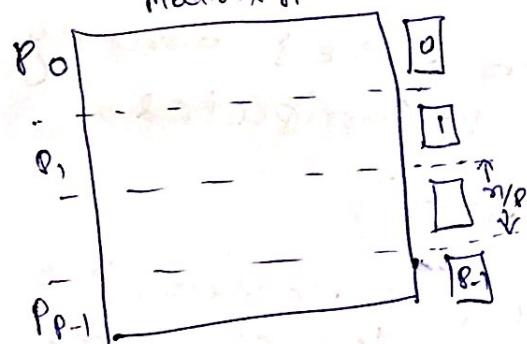
$$C_2 = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix} + \begin{bmatrix} 62 & 19 \\ 42 & 20 \end{bmatrix} = \begin{bmatrix} 82 & 55 \\ 57 & 83 \end{bmatrix}$$

$$C_3 = \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix} + \begin{bmatrix} 0 & -12 \\ 48 & -46 \end{bmatrix} = \begin{bmatrix} 30 & 25 \\ 86 & 30 \end{bmatrix}$$

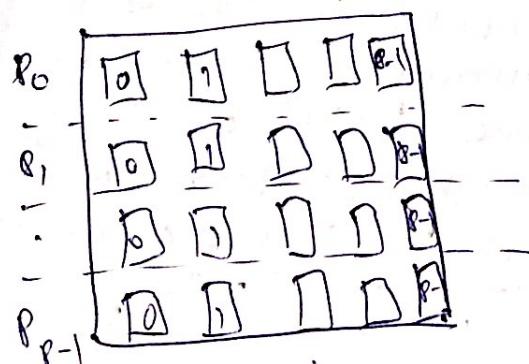
$$C = \begin{bmatrix} C_0 & C_1 \\ C_2 & C_3 \end{bmatrix} = \begin{bmatrix} 338 & 52 & 26 & -14 \\ 53 & 64 & +2 & 57 \\ 82 & 55 & 30 & 25 \\ 57 & 83 & 86 & -7 \end{bmatrix}$$

MATRIX VECTOR MULTIPLICATION

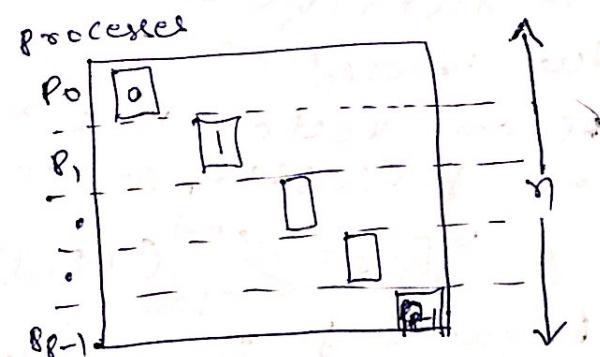
multiplication of an $n \times n$ matrix with an $n \times 1$ vector using row wise block 1-D partitioning
for one row-per-process case, $p=8$



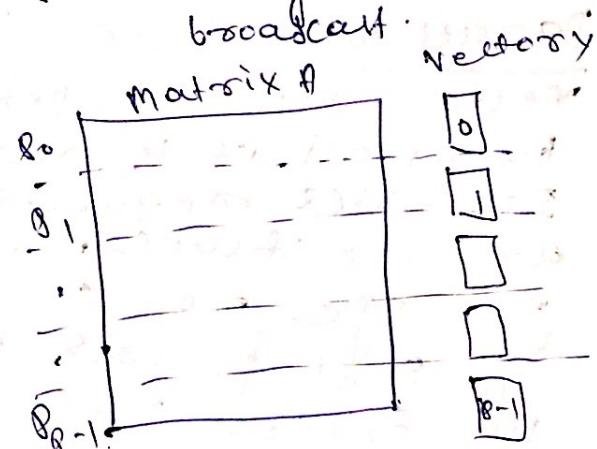
(a) Initial partitioning of the matrix and the starting vector X



(c) Entire vector distribution to each process after the broadcast



(b) Distribution of full vector among all the processes by auto-all broadcast



final distribution of the matrix and the result vector Y

one row per process

Consider the case in which the $n \times n$ matrix is partitioned among n processes so that each process stores one complete row of the matrix. Then $n \times 1$ vector X is distributed such that each process owns one of the elements. The initial distribution of the matrix and the vectors for rowwise block 1-D partitioning is shown in fig (a).

→ Process P_i initially owns $X[i]$ and $A[i,0], A[i,1], \dots, A[i,n-1]$ and is responsible for computing $Y[i]$.

→ Vector X is multiplied with each row of the matrix. So every process needs the entire ~~vector~~ vector. Since each process starts ~~with~~ with only one element of X , an all-to-all broadcast is required to distribute all the elements to all the processes.

After vector X is distributed among all processes, process P_i computes

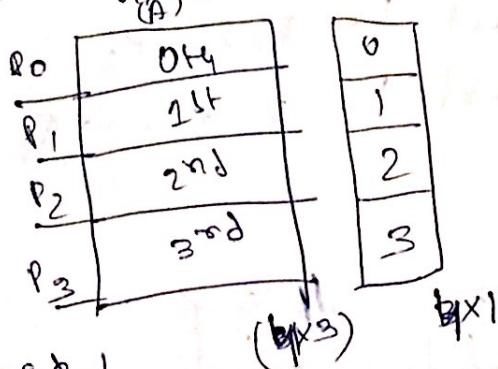
$$Y[i] = \sum_{j=0}^{n-1} (A[i,j] \times X[j])$$

Figure (d) shows, the result vector Y is stored exactly the way the starting vector X was stored.

parallel run time — starting with one vector element per process, an all-to-all broadcast of the vector elements among n processes requires time $\Theta(n)$ of any architecture. The multiplication of a single row of A with X is also performed by each process in time $\Theta(n)$. Thus the entire procedure is completed by n processes in time $\Theta(n)$ resulting in a process-time product of $\Theta(n^2)$.

The parallel algorithm is cost optimal because the complexity of the serial algorithm is $\Theta(n^2)$.

$P_2 X$



Step-1

no of rows M , each row of a matrix A carries one element of vector X that means 0th row of A is given to P_0 and is assigned to 0th element of vector

similarly 1st row of matrix A is given to P_1 and is assigned to the 1st element of

P_1 and is assigned

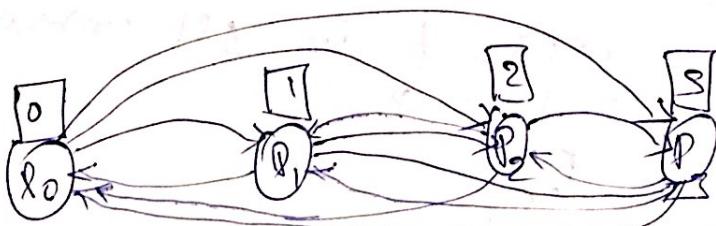
Vector X in the manner 2nd row of matrix A is given to P_2 process and is assigned to 2nd element of vector X

3rd row of matrix A is given to process P_3 and assigned 3rd element of vector X .

Step-2

but according to our matrix multiplication one row will be multiply with all the vector elements of X but P_0 is holding only 0th element of X , P_1 is holding only 1st element of X , P_2 is holding only 2nd element of X and P_3 is holding only 3rd element of X

P_0 P_1 P_2
 P_3 is holding only 3rd element of X
 so broadcast is necessary (all to all broadcast.)



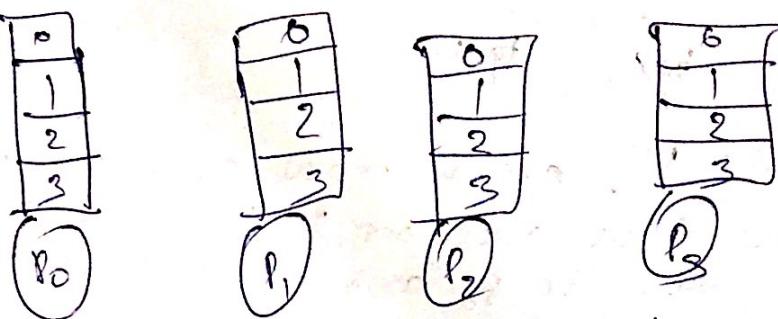
broadcast P_0 element of X to P_1, P_2 and P_3

P_1 element is broadcast to P_0, P_1, P_2 .

P_2 element is broadcast to P_1, P_0, P_3

P_3 element is broadcast to $P_0, P_1, \text{and } P_3$

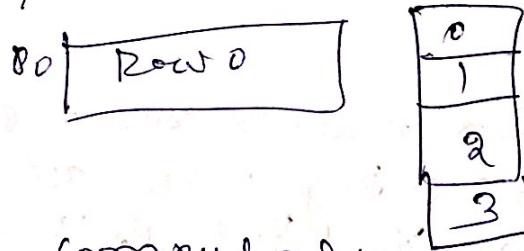
Step-IV



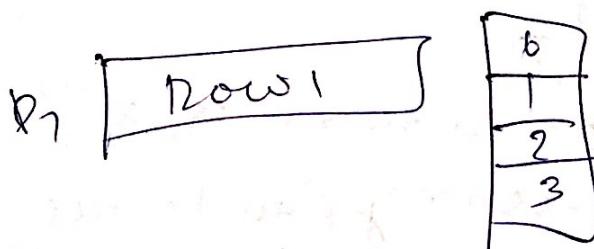
So P_0 process is assigned to 0th row of matrix X and now every process has a unique vector (complete vector).

so P_0 is holding the row as well as complete vector. Similarly all P_1, P_2 and P_3 also contain their complete vector individually.

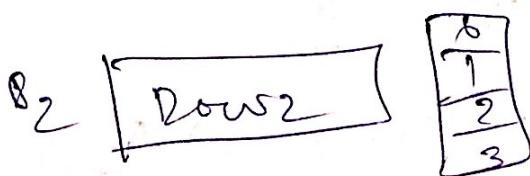
Now multiply process P_0 's row with vector / row₀ with vector.



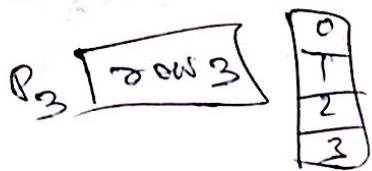
After computation store the result in 0th index of R.



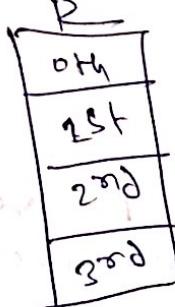
After computation row1 with vector store the result in 1st index of resultant vector R.



after computation row 2 with vector
store the resultant in 2nd index of
resultant vector R



after computation row 3 with vector
store the resultant in 3rd index of
Resultant Vector R



Resultant Vector

Vector-matrix multiplication using 2D partitioning (Dense matrix algorithm)

it consists of a $n \times n$ matrix multiplied
with vectors ($n \times 1$)

→ it consists of (matrix) rows and columns

Step-1 → $A \times b$ (vector) \rightarrow n^2 elements

matrix A			
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36
37	38	39	40

0
1
2
3

- 0-th index

- 1-th index

- 2nd index

- 3-index

$(n \times n) \times (n \times 1)$

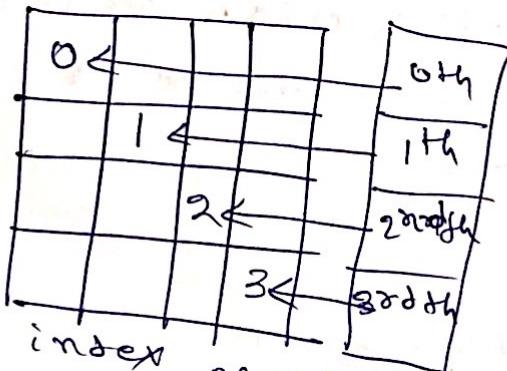
→ one element per process for computation
purpose

→ Total no. of processes $P = n^2$ elements.
for each element one process is assigned

Step-11

~~put 0th element, 1th element, 2th element
and 3th element~~

Put 0th index, 1th index, 2th index and 3th index of vector (x) below on given picture



all index elements are present diagonally in the one-to-one communication above figure

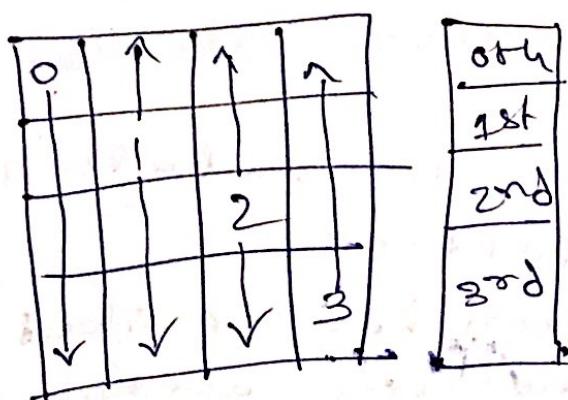
Step - III

apply one to all communication because

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

In the above picture 0th index element of vector is multiplied with 0th column of the matrix. it will not interact to other. So computation will only occur 0th index element of vector with 0th column and similarly 1st index element of vector will interact with 1st column
→ 2nd index element of vector will interact with 2nd column of matrix
→ 3rd index element of vector will interact with 3rd column of matrix
→ 4th index element of vector will interact with 4th column of matrix.

so one to all broadcast is required
so broadcast to one element to all element in column wise



So after broadcasting 0th index value of vector will reach for all processes in the 0th column.

→ similarly 1st index value of vector will reach for all processes in the 1st column.

→ 2nd index value of vector will reach for all processes in the 2nd column.
 → 3rd index value of vector will reach for all processes in the 3rd column.

Step-IV

After computation because every cell contains element and each cell assigns one process each in ~~row~~ column wise so in the given picture above 0th column contains 4 processes and four elements are present so it will multiply with 0th index element of vector X similarly 1st index element multiplying with 1st column.

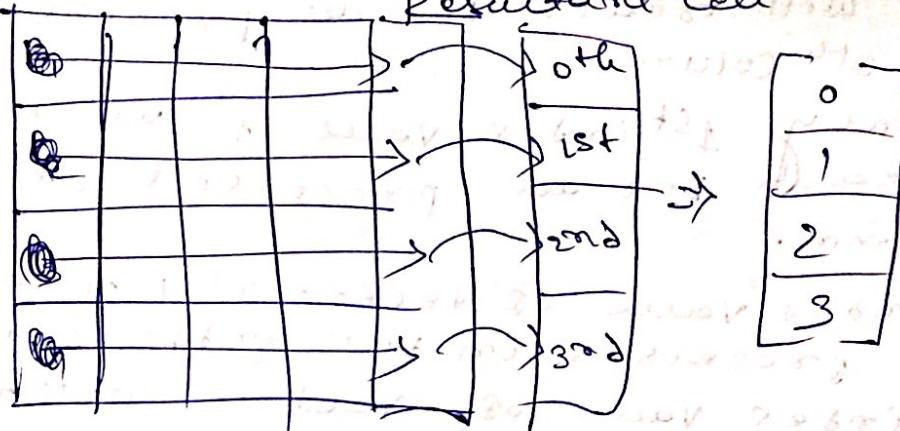
computation occurs 2nd index element of vector with 2nd column elements

computation occurs 3rd index element of vector with 3rd column elements.

Step-V

after performing the computation apply all to one reduction when all the computation has finished every cell contains the computation result in row and column format.

Resultant cell



Accumulate every cell element row wise and adding the elements row wise cell element and stores it to the result and returns 0th index, 1th index (second 1st row elements), 2nd index (2nd row cell elements), and 3rd index (3rd row cell elements).

$$\text{Complexity} = \Theta(n^2 \log n)$$

If we apply sequentially for matrix multiplication complexity = $\Theta(n^2)$ so 2-D technique is not cost optimal.

So transpose both the matrix and do the same thing.

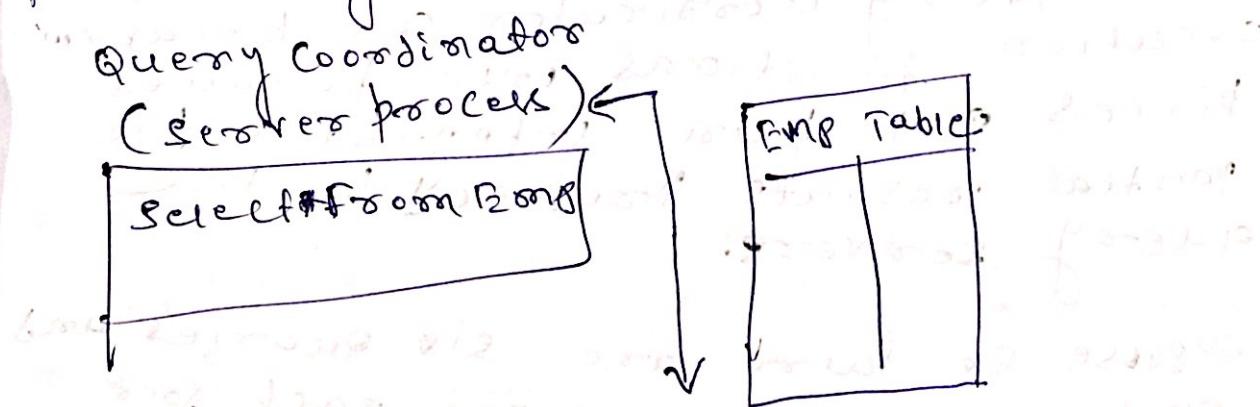
So transpose both the matrix and do the same thing.

Database Query processing in parallel computers

The processing of a SQL statement is always performed by a single server process. → with the parallel query feature, multiple processes can work together simultaneously to process a single SQL statement. This capability is called parallel query processing. - by deriving the work necessary to process a statement among multiple server processes, parallel query process architecture.

Without the parallel query feature, a server process performs all necessary processing for the execution of a SQL statement.

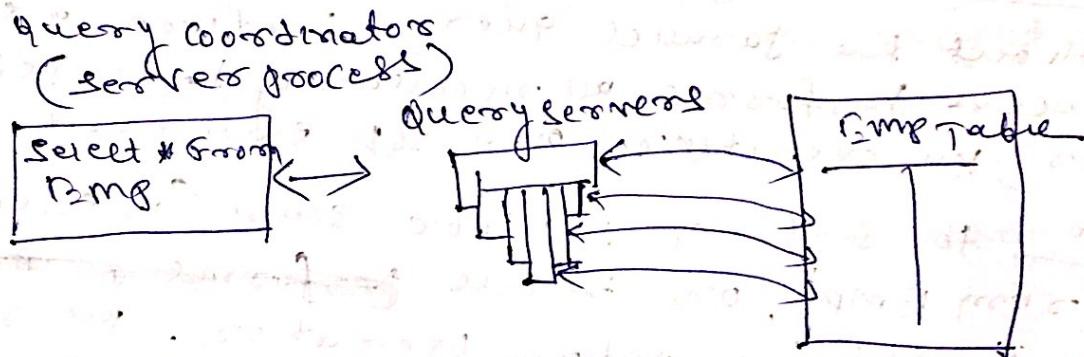
Ex to perform full table scan (Ex select * from BMP), one process ~~performs a~~ performs the entire operation. The following figure illustrates a server process performing a full table scan.



(full table scan without the parallel query feature)

The parallel query feature allows certain operations (for example, full table scan or sorts) to be performed in parallel by multiple query server processes, distributes the execution of statements to several query servers and coordinates the results from all of the servers to send back to the user.

- The following figure illustrates several query servers process simultaneously performing a partial scan of the EMP table. The result are then sent back to the query co-ordinator, which assembles the pieces into the desired full table scan.



Multiple query servers performing a full table scan in parallel.

- The query coordinator can breakdown execution functions into parallel pieces and then integrate the partial results produced by the query servers.

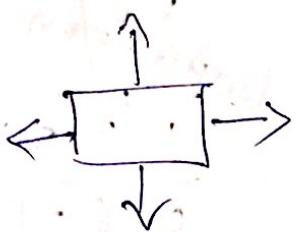
Suppose ex there are six queries and each query takes 3 sec each for execution.

So total time it take to execute = $6 \times 3 = 18$ seconds for all Queries in sequentially.

but is few queries and subdivided into different parts and each sub part are assigned to different processes. So execution of all the queries will be completed within the 3 seconds.

15 PUZZLE PROBLEM

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



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

Goal arrangement

An arrangement

We will move the blank plate after from UP, down, left and right position. After this we calculate the error means which arrangement gives minimum error. We will branch again to find the goal arrangement and other we will bound the steps branch the step which gives the minimum error.

Ex minimum error

Description

In step 1 - we move left; right, down, up
 In step 2 - after calculating the error
 In step 3 - node 9 has min error (2). So branch node 9 and found node 1, node 2 and node 4
 In step 4 - node 5 has min error (1) so branch node 5 and found node 6 and node 7

root node

Step-1

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



Step-3

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	15	17
13	14	/ / /	12

1	2	3	4
5	6	7	8
9	10	15	17
13	14	15	12

Step-4

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

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

goal node (arrangement)

in step - 4 mode 9 is the resultant
goal.
parallel
Discrete event simulation

- A simulation model can be viewed as a representation of the physical system under simulation.
- Discrete event simulation (DES) are characterized by discrete-state models (as opposed to continuous state models) and the event driven approach.
 - in continuous state model, or continuous simulation, the state of the simulation changes continuously over time while in discrete state model, or discrete simulation, the state of the simulation is instantaneous and occurs at discrete point in simulation time (the discrete point in simulation time the discrete applies only to the state changes but not to value of the simulated time)

Sequential simulation —

- All sequential DES have a common structure
- The set of future events is ~~scheduled~~
is maintained in a global event list
- These events are to be scheduled by an event scheduler in a non-decreasing time stamping order, one after another
- After processing an event, the simulation clock recording the current time is then advanced to the time of the next earliest occurring event.

parallel simulation

- decomposing a simulation for processing on multiple processes
- ### principles of parallel simulation

In PDES, a model generally comprised N_A processes (L_P) $L_P = L_{P1}, \dots, L_{PN-1}$ which interact among themselves by sending time stamped event messages. A link exists from L_{Pi} to L_{Pj} if L_{Pi} may send messages to L_{Pj} .

- each message is a tuple (E_k, T_k) consisting of an event E_k and its associated time stamp T_k .
- each L_P , L_{Pi} is associated with a local clock, or local virtual time (LVT), clock; is automatically advanced to T_k .

Let E_1 and E_2 denote two events with timestamps T_1 and T_2 respectively. Assume that $T_1 < T_2$. If E_2 depends on E_1 , then there is a causality constraint over the order of execution of E_1 and E_2 . For instant, if E_1 causes a change of state variable which will be referenced by E_2 , then E_1 must be executed before E_2 or a causality error will occur.

- it has been seen that a parallel simulation obeys the causality constraint if and only if every LP processes events in nondecreasing timestamp order - a condition known as a local causality constraint. It is important to note that local causality is a sufficient condition but not a necessary condition.
- violating causality constraint means that the future can affect the past.
- This can result in anomalous behavior and consequently incorrect simulation.
- It is responsibility of the synchronization mechanism to ensure proper and correct interactions among the LPs.
- * If two events in the same LP have different time stamps but there is no (direct or indirect) dependency between them, then they can be executed in parallel without causing causality errors.

MODULE - I

Hyper Quick Sort

Sorting is a process of arranging elements in a group in a particular order that is ascending order, descending order, alphabetic order etc.

- Enumeration Sort
- Odd-Even Transposition Sort
- Parallel merge sort
- Hyper quick sort

Sorting a list of elements is a very common operation. A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore parallel algorithms are used in sorting.

Enumeration Sort

Enumeration sort is a method of arranging all elements in a list by finding the final position of each elements in a sorted list. It is done by comparing each element with all other elements and finding the number of elements having smaller value.

These are any two elements, a_i and a_j any one of the following must be true:

- $a_i < a_j$
- $a_i > a_j$
- $a_i = a_j$

Algorithm

procedure ENUM-SORTING(n)

begin

for each process P_1, S do

$C[j] := 0;$

for each process P_i, j do

if ($A[i] < A[j]$ or $A[i] = A[j]$ and $i < j$)

$c[i] = 0;$
 else
 $c[i] := 0;$
 for each process $P_1, j \neq 0$
 $A[c[i]] := A[i];$
 end ENUM_Sorting

Ex

$A = \{2, 5, 0, 4\}$ partition

2	$\boxed{2, 2}$	$\boxed{2, 5}$	$\boxed{2, 0}$	$\boxed{2, 4}$	\circ	$0+0+1+0=1$
	$P(1,1)$	$P(1,2)$	$P(1,3)$	$P(1,4)$		
5	$\boxed{5, 2}$	$\boxed{5, 5}$	$\boxed{5, 0}$	$\boxed{5, 4}$	\circ	$-1+0+1+1=0$
	$P(2,1)$	$P(2,2)$	$P(2,3)$	$P(2,4)$		
0	$\boxed{0, 2}$	$\boxed{0, 5}$	$\boxed{0, 0}$	$\boxed{0, 4}$	\circ	$-0+0+0+0=0$
	$P(2,1)$	$P(3,2)$	$P(3,3)$	$P(3,4)$		
4	$\boxed{4, 2}$	$\boxed{4, 5}$	$\boxed{4, 0}$	$\boxed{4, 4}$	\circ	$-1+0+1+0=2$
	$P(4,1)$	$P(4,2)$	$P(4,3)$	$P(4,4)$		

consider only if $P(i,j) = 1$ if $i > j$ then place 1 otherwise if $i = j$ and $i < j$ then place 0

Ex $\boxed{2, 2}$ in this case $i = j$ place 0

$\boxed{2, 5}$ - $i < j$ here is also placed 0

$\boxed{2, 0}$ - in this case $i > j$ placed 1

$\boxed{2, 4}$ - here in this case $i < j$ placed 0

$\boxed{5, 2}$ - here in this case $i > j$ placed 0

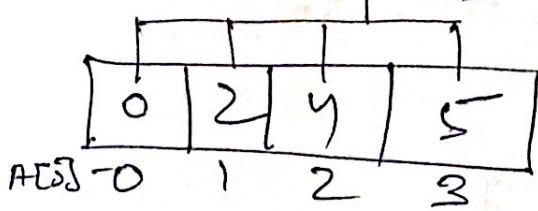
Step-11

change the position such that element 4 is placed in 2nd free position in tree list.

0 is placed 0th element in tree list

2 is placed 1st position in tree list

5 is placed in the 3rd element
in the list $c[j]$



$$A[j] = A[c[j]] \text{ means}$$

$$A[0] = A[c[0]]$$

$$\Rightarrow A[0] = A[0] = 0$$

$$A[1] = A[c[1]]$$

= 2

$$A[2] = A[c[2]] = 4$$

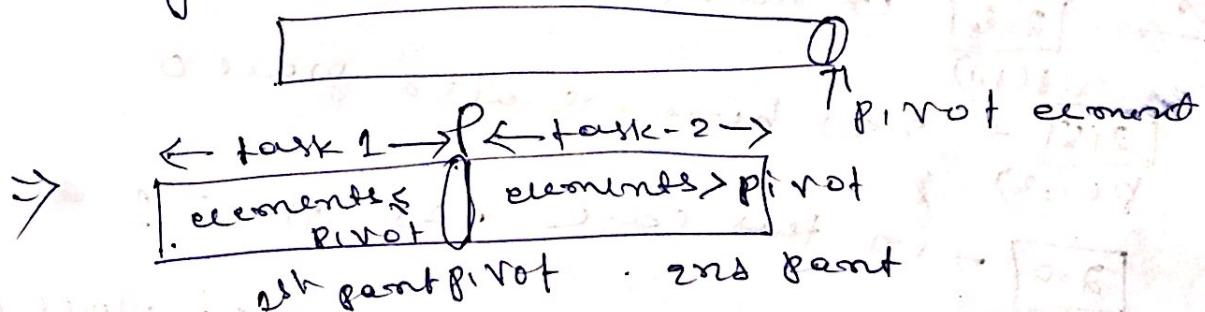
$$A[3] = A[c[3]] = 5$$

Parallel

Quick Sort

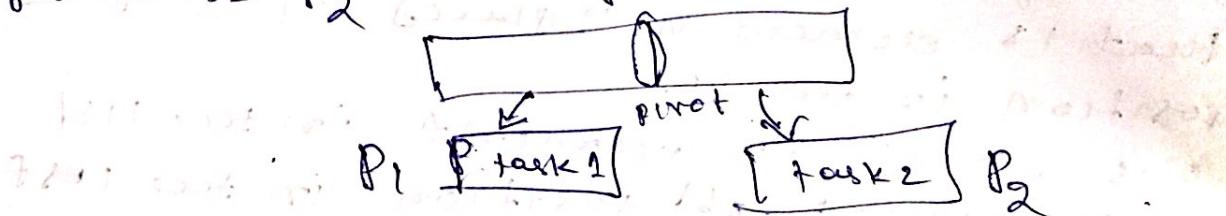
Step-1.

Select last element in the array as pivot element and divide the array.



Processor P is partitioned the array into two parts. After partitioning the two tasks are independent to each other.

assign the sub tasks for process P_1 and process P_2

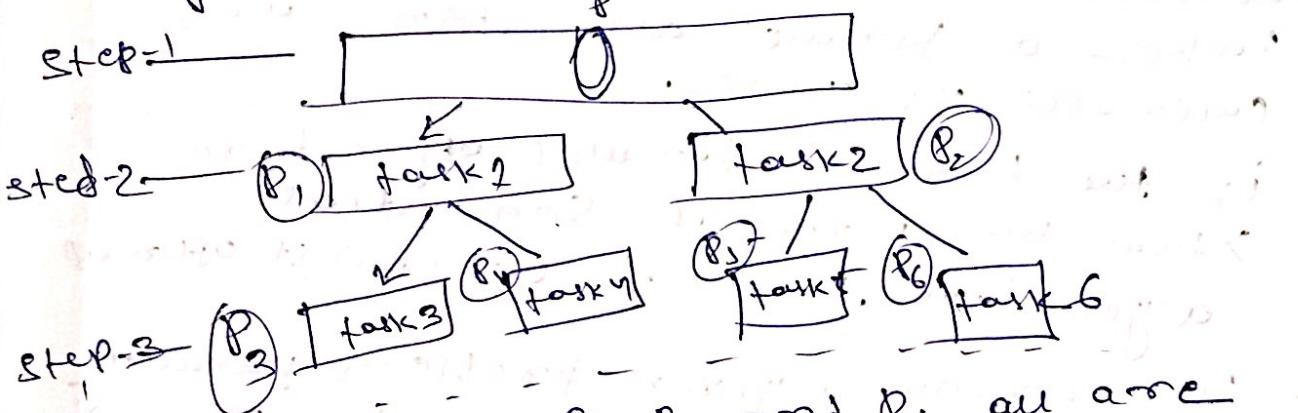


so P₁ process will execute task₁ and process P₂ will execute task₂ independently parallelly.

for task₁ select pivot element and again divide the task₁ for two subtask (task₃, task₄) and similarly we will also divide the task₂ into subtask (task₅ and task₆) and assign process to task₃ and task₄.

process to task₅ and task₆

assign process to task₅ and task₆



step-n - processes P₃, P₄, P₅ and P₆ all are assigned task₃, task₄, task₅ and task₆ individually and parallelly. so task₃, task₄, task₅ and task₆ are executed parallelly at the same time.

- the process will continue until every process contains single single element that means single single element belongs to each process.

finally a shortest list of array will generate.

→ it has been seen that at every step a single process divides the task into sub tasks (P-single process divides the sub tasks from task₁ to task₂ and task₃ to task₆). P₁ divides the subtask₁ into task₃ and task₄.

Similarly, P_2 also divides the task into tasks 5 and task 6.

Similarly process P_3, P_4, P_5 and P_6 divides the task accordingly.

So it is not cost optimal.

* Drawback is some of the steps are belong to sequentially, so it is not cost optimal.

Cost optimal means if we take a best sequential algorithm and calculate its ~~cost~~ time and we take a parallel algorithm and calculate its time.

If the time of parallel algorithm

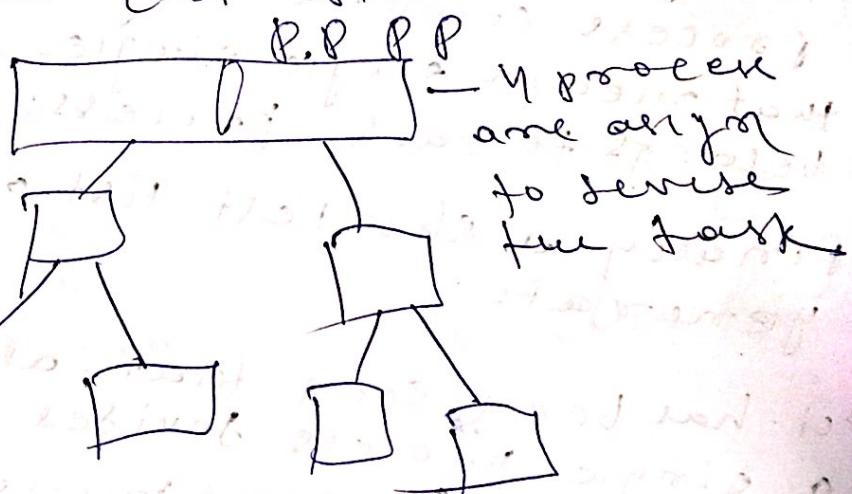
> than the time of sequential

algorithm then it is not cost optimal.

* For solving a given problem parallelly it takes more time than solving the same problem sequentially we call it that it is not cost optimal.

How to make cost optimal:

* If we assign more than one process for partitioning the task then we make it as cost optimal.



If partitioning step 1 is performed using $O(1)$, using $O(n)$ processes.

it is possible to get parallel run time as $O(\log n)$

* we will parallelize partitioning step by using n processes

Hyper Quick Sort

Hyper Quick Sort is an implementation of Quick Sort on hypercube. The steps are as follows.

- divide the unsorted list among each node
- sort each node locally
- from node 0, broadcast the median value.
- split each list locally, then exchange the halves across the highest dimension
- repeat step-3 and step-4 in parallel until the dimension reaches zero

Algorithm

procedure HYPERQUICKSORT(B, N)

begin

 if $i := \text{processors label};$

 for $i := 1$ to d do

 begin

$x := \text{pivot};$

 partition B into B_1 and B_2 such that

 that $B_1 \leq x < B_2;$

 if i^{th} bit is 0 then

 begin

 send B_2 to process along the i^{th}

 Communication Link

C_i^1 = Subsequence received along the
 i^{th} communication link;
 $B_1^1 \neq B_1 V C$;
 end if
 else
 send B_1 to process along the i^{th}
 communication link;
 C_i^2 = Subsequence received along the
 i^{th} communication link;
 $B_1^2 = B_2 V C$;
 end else
 end for
 send B using sequential form;
 end HYBRICKSORT;

Ex
 Given an array of elements given
 below by using parallel processing
 solve the problem

P_0	\longleftrightarrow	P_1							
9	7	5	11	12	2	14	3	10	6

suppose there are two processors
 P_0 and P_1 , assign processor P_0 to 1st
 part and P_1 to second part
 suppose 6 is pivot element

5	9	7	11	12	2	3	6	14	10
---	---	---	----	----	---	---	---	----	----

Now P_0 has 6th element as pivot
 and P_1 has 5th element as pivot
 send 6 to P_0 and 5 to P_1

After swap sorting of 6th element
 and 5th element

Odd-even Transposition Sort

- Odd-even Transposition sort is based on the bubble sort technique. It compares two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order series.
- The ~~positive~~ opposite case applies for a descending order series.
- odd-even transposition sort operates in two phases - odd phase and even phase.
- in both the phases, process exchange numbers with their adjacent numbers in the right.

Ex

1 9 <hr/> 7 <hr/> 3 <hr/> 8 <hr/> 5 <hr/> 6 <hr/> 4 <hr/> 1	2 3 <hr/> 8 <hr/> 5 <hr/> 6 <hr/> 1 <hr/> 4 <hr/> 7 <hr/> 8 <hr/> 6 <hr/> 1 <hr/> 7 <hr/> 4 <hr/> 8 <hr/> 6 <hr/> 1 <hr/> 8 <hr/> 7 <hr/> 4 <hr/> 6 <hr/> 1 <hr/> 3 <hr/> 5 <hr/> 1 <hr/> 7 <hr/> 4 <hr/> 9 <hr/> 6 <hr/> 8 <hr/> 1 <hr/> 5 <hr/> 4 <hr/> 7 <hr/> 6 <hr/> 9 <hr/> 1 <hr/> 3 <hr/> 4 <hr/> 5 <hr/> 6 <hr/> 7 <hr/> 8 <hr/> 9	unsorted	7 8	9 1 - phase 1 (odd)
				4 - phase 2 (even)
				7 9 3 8 5 6 1 4 - phase 3 (odd)
				7 3 9 8 1 6 4 - phase 4 (even)
				3 7 8 9 1 8 4 6 - phase 5 (odd)
				3 8 7 1 9 4 8 6 - phase 6 (even)
				(3) 8 1 7 4 9 6 8 - phase 7 (odd)
				3 5 1 7 4 9 6 8 - sorted

Bx

1	2	3	4
3 2	3 8	5 6	1 1

Suppose there are eight processes and there are and each process holds a single element.

In the problem eight processes are there and each holding one elements in the array.

P_1 - holding 1st element in the array

P_2 - " 2nd "

P_3 - " 3rd "

P_4 - " 4th "

P_5 - holding 5th "

P_6 - holding 6th "

We will give the name odd/even by seeing the index label.

as P_1 process P_1 if we consider

P_1 - index is odd - so we will give label odd

Similarly P_2 - the index is even

so we will give label even

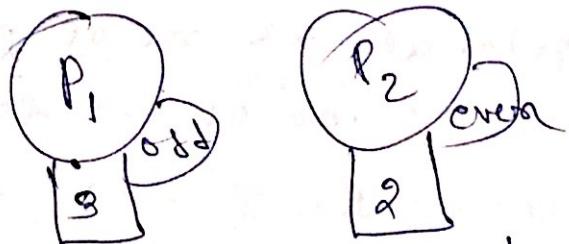
similarly other will be calculated.

If we consider P_1 process holds a

value of 3

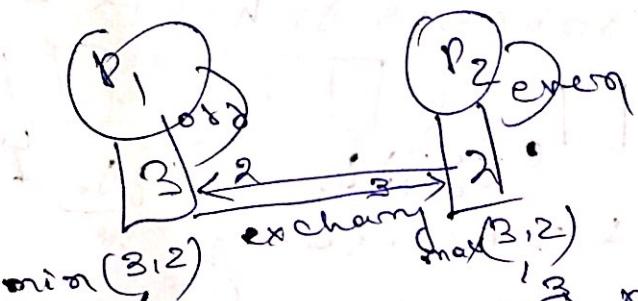
and P_2 process holds a value of 2

→ Checking the spear is odd fall or even fall



you have to check the 1st element
process P_1 - holding the 1st index val.
of array and its label is odd so it
is formed in odd cell [space is
formed in odd cell]

calculating the even cell is same as
that of above
- check the 1st element and if the
index is even and label is even,
space is formed in even cell.



* two processes P_1 and P_2 will
exchange their value [they will communicate
to each other]

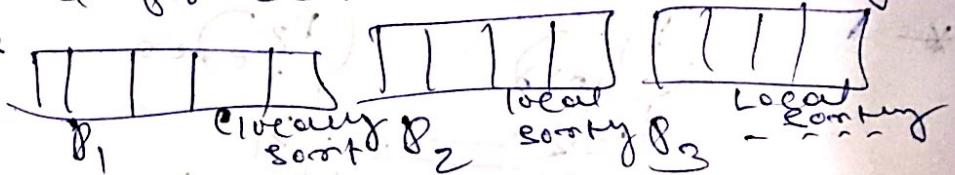
- after communication P_1 contains
value (3,2) and P_2 contains value (2,3)
- then each process contains a
pair of values
- in process P_1 calculate the min value
in pair of elements
- in process P_2 - calculate the maximum
value in pair of elements
- so sorted element is formed as
 $\boxed{2,3}$

it is not cost optimal as no of process
is not less than n (no of elements)
because here element = 8 and process
= 8
Cost optimality

if no of process < no of element
heat means each process
holds = $\frac{n}{p}$ no of elements

Time complexity
if each process contains a array
of elements then local sorting is
done with in a one process.

Ex - suppose a process P_1 holds array
of elements



local sorting is necessary

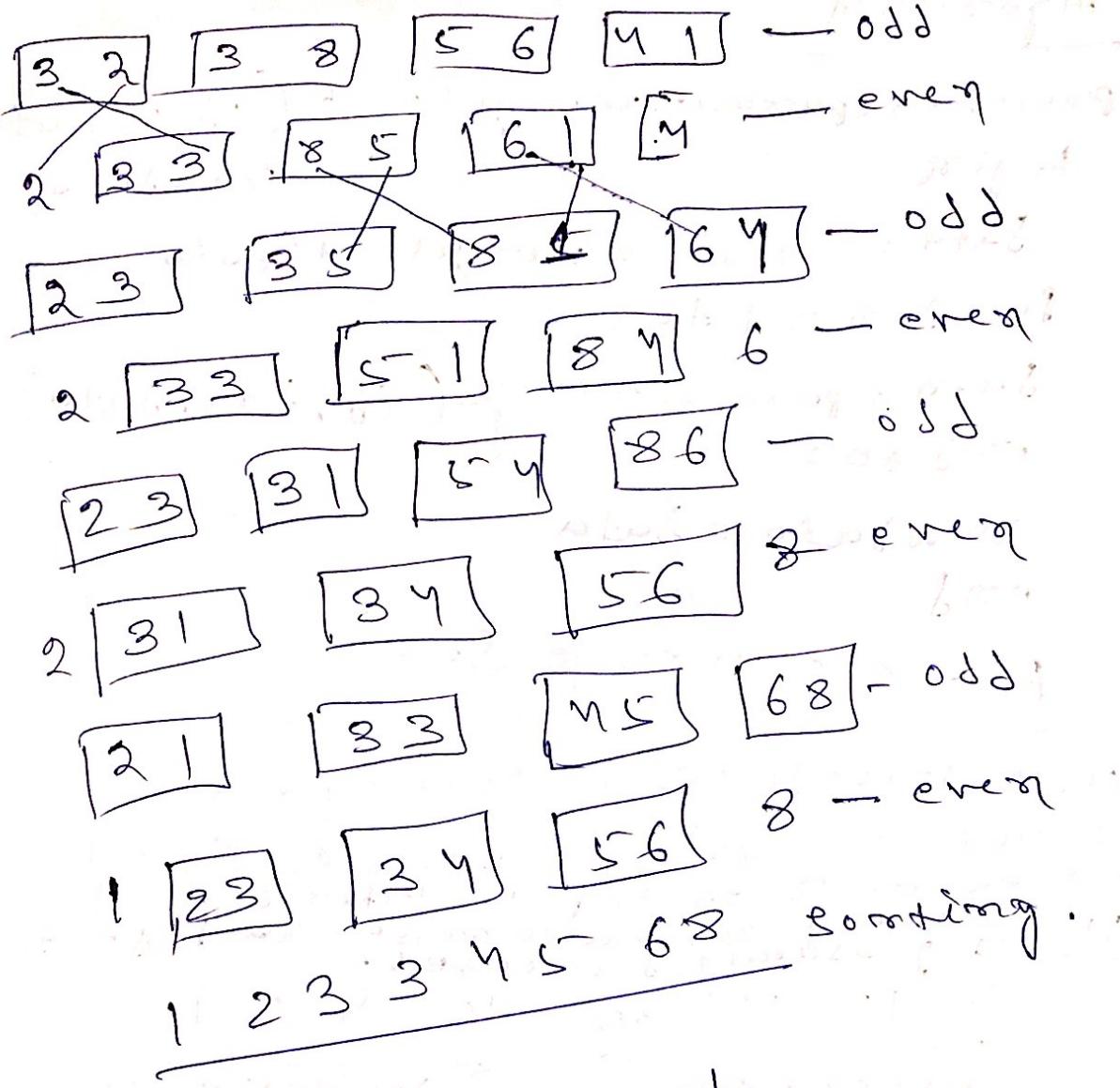
Time it take for local sorting

$$T_p = \Theta(n/p \log n/p)$$

II - Communication for exchanging
of data - $\Theta(n)$

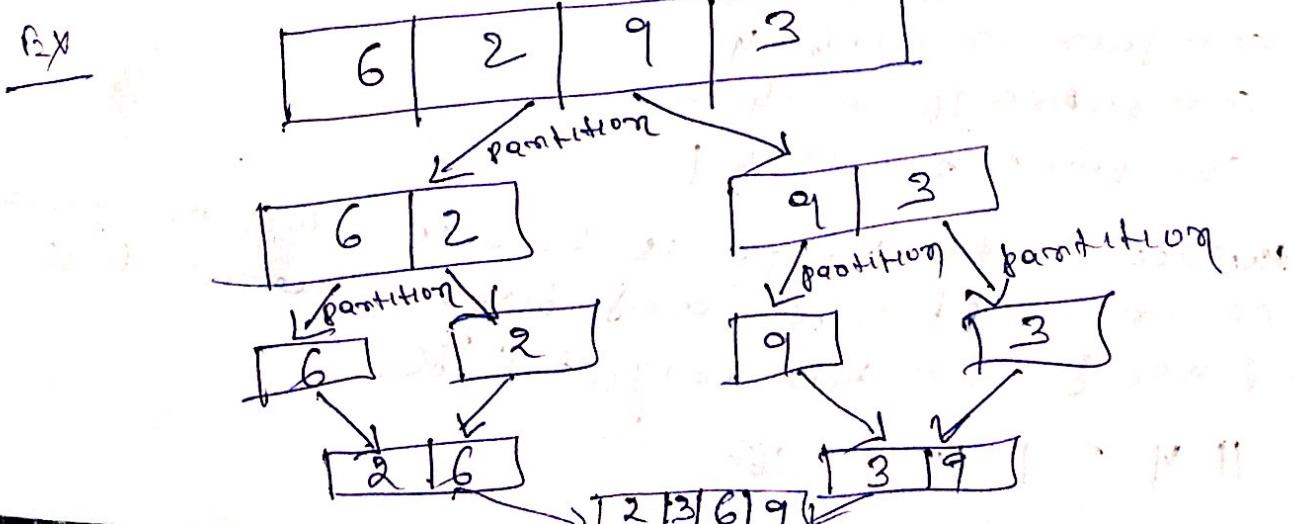
III - comparison - $\Theta(n)$

Total time complexity - $T_p = \Theta(n/p \log n/p + \Theta(n) + \Theta(n))$



parallel merge sort

Merge sort first divides the unsorted list into smallest possible sub-list compared it with the adjacent list, and merges it in a sorted order. It implements it in a sorted order by following the parallel and conquer algorithm.



Algorithm

```

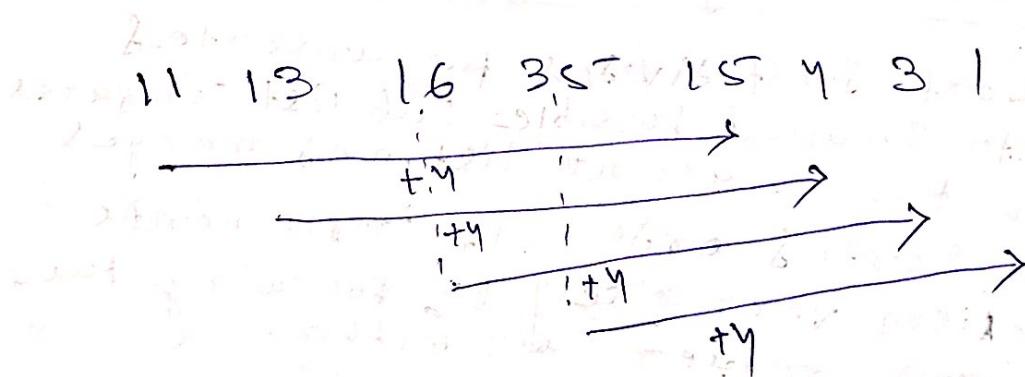
procedure parallelmergeSort (id, n, data,
begin
    data = sequentialmergeSort (data)
    for dim = 1 to n
        data = parallelmerge (id, dim, data)
    end for
    newdata = data
end

```

Bitonic merge Sort

It is procedure of sorting a bitonic sequence using bitonic sort. A sequence is said to be bitonic if it gradually increases and after half position it gradually decreases.

11 13 16 35 : 15 4 3 1
 ↓↓↓↓ → : ← ↓↓↓↓
 ascending order descending order



compare 11 with 15

compare 13 with 4

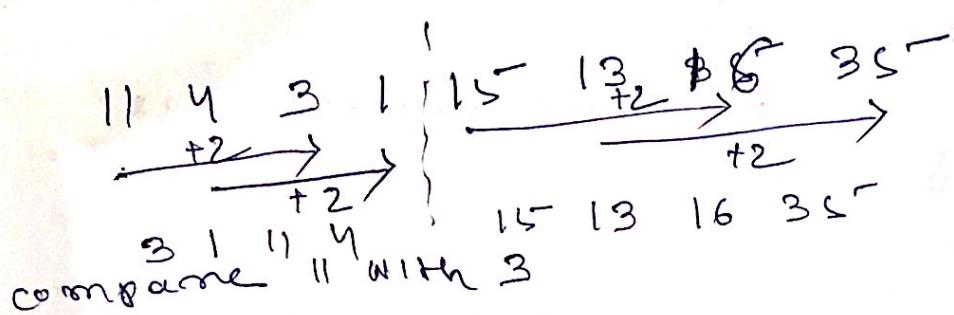
compare 16 with 3

compare 35 with 1

the element which is smaller we will place it in left side and bigger element is placed in the right side

11 4 3 1 15 13 16 35

then compare +2 (last two elements)



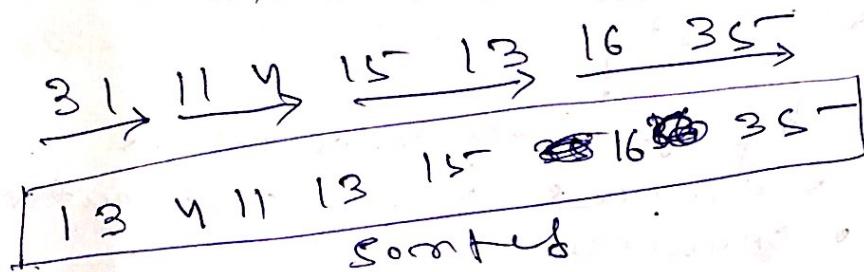
compare 4 with 15

in the part 2 compare 15 with 16
and compare 13 with 35

$\rightarrow 3 \ 1 \ 11 \ 4 \ 15 \ 13 \ 16 \ 35$

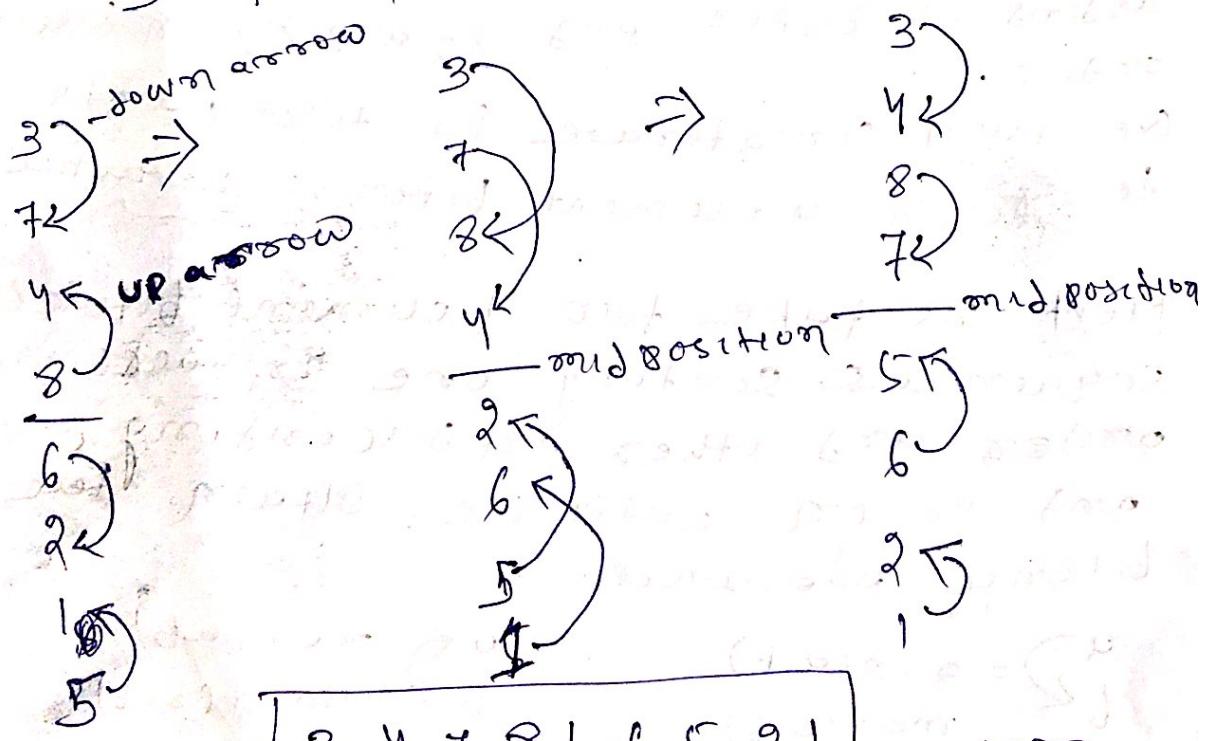
\rightarrow Comparison with +1 element

means compare 3 with 1, 11 with 4,
15 with 13, and 16 with 35



$BX-2$

$\underline{\underline{3 \ 7 \ 4 \ 8 \ 6 \ 2 \ 1 \ 5}}$



- * up arrow - bigger element is replaced to smaller one
- * down arrow - smaller element is replaced.

Description

an array $arr[0 \dots n-1]$ is bitonic if there exists an index i where $0 \leq i \leq n-1$ such that

$$x_0 \leq x_1 \leq \dots \leq x_i \text{ and } x_i \geq x_{i+1} \geq \dots \geq x_{n-1}$$

A sequence, sorted in increasing order is considered bitonic with the decreasing part an empty.

Similarly decreasing order sequence is considered bitonic with the increasing part as empty.

We start by forming 4-element bitonic sequences from consecutive 2-element sequences.

Consider 4-elements in sequence x_0, x_1, x_2, x_3 . we sort x_0 with x_1 in ascending order and x_2 and x_3 in descending order.

we then concatenate the two parts to form 4 element bitonic sequence.

Next we take two 4 element bitonic sequences, sorting one in ascending order and other in descending order and so on, until we obtain the bitonic sequence.

$$\begin{array}{l} a = \max(a, b) \\ b = \min(a, b) \end{array}$$

$$\begin{array}{l} a = \max(a, b) \\ b = \min(a, b) \end{array}$$

Anaylsis of bitonic sort

To form a sorted sequence of ~~length n~~ length n from two sorted sequences of length $n/2$, $\log(n)$ comparisons are required.

$$\rightarrow T(n) = \log(n) + T(n/2)$$

The solution of this recurrence equation is

$$T(n) = \log_2 \cdot (\log(n) + 1)/2$$

Worst case performance $O(\log^2(n))$ - parallel time

Best-case performance $O(\log(n))$ - parallel time

Average performance $O(\log^2(n))$ - parallel time

Worst-case space complexity - $O(n \log^2(n))$ - parallel

EREW MODEL

\rightarrow Let x

$$\text{Let } k = 2^{(1-x)}$$

BRew SORT

procedure R=REWSORT(S)

if $|S| \leq K$

then Quicksort(S)

else (1)

for $i = 1$ to $K-1$ do

PARALLELSELECT ($S, \Gamma_i(S)/K$)

{ obtain m_i }

end for

(2) $S_1 \leftarrow \{s \in S : s \leq m_1\}$

(3) for $i = 2$ to $K-1$ do

(5) For $i=1$ to $k/2$ do in parallel
 BERN SORT(s_i)
 end for

(6) For $i=(k/2)+1$ to k do in parallel
 BERN SORT(s_i)
 end for
 end if \square

B20

A sequence of integers

$S = \{s_1, s_2, \dots, s_n\}$ and an integer k
 are given and it is required to
 find the k th element

Algorithm

- p no of processors are available
- each processor will receive
 $[S]^{\frac{1}{p}}$ elements such that ($p = n^{\frac{1}{k}}$)
- find median of each subsequence
 $[s_{i/2}]$ and place it in M .
- Now find median of M . treat it as
 $[M_{1/2}]$
- Sequence divided into L, B, U
 where $L = \text{less}, B = \text{equal}, U = \text{greater}$

$$L = \{s_i \in S; s_i < M\}$$

$$B = \{s_i \in S; s_i = M\}$$

$$U = \{s_i \in S; s_i > M\}$$

Bx

3	14	16	20	8	31	14	16	12	22	33	25	7	19	13	14	19	15	24	26	3	36	11	27	22	33	25
---	----	----	----	---	----	----	----	----	----	----	----	---	----	----	----	----	----	----	----	---	----	----	----	----	----	----

5	14	16	20	8	31	22	33	1	14	13	14	15	16	17	18	12	11	26	18	31	16	25	14	27	33	26
---	----	----	----	---	----	----	----	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

given $k^{th} = 21^{\text{st}}$ position - find out the
 21^{st} position in the array

133

$$k^{th} = 21^{\text{st}} \quad \phi = 5 \quad n = 29$$

$$[5]^n \Rightarrow (29)^n$$

$$\rho = n^{1-n}$$

$$5 = [29]^{1-n}$$

$$\log 5 = (1-n) \log 29$$

$$\Rightarrow n = 0.5220 \rightarrow (1)$$

put the value of n in $[5]^n$

$$[29]^{0.5220} = 5.75 \approx 6$$

so every processor will assign 6 elements each.

P2

P_1	3	14	16	20	8	31
-------	---	----	----	----	---	----

1	14	16	12	22	33
---	----	----	----	----	----

P_3	2	5	7	10	13	14
-------	---	---	---	----	----	----

8	9	18	25	26	34	36
---	---	----	----	----	----	----

P_5	11	27	32	33	35	36
-------	----	----	----	----	----	----

5 element

Step-1)
 sorting each processor elements and
 calculate the median value of each processor

after sorting

2	8	14	16	20	31
1	2	3	4	5	6

median is $M_1 = \frac{f_1}{2} = 3^{\text{rd}}$ position's element

$$M_1 = 14$$

P₂

1	3	9	12	22	33
1	2	3	4	5	6

$M_2 = \frac{f_2}{2} = 8^{\text{th}}$ element

$$M_2 = 9$$

2	5	7	10	13	14
1	2	3	4	5	6

$$M_3 = 7$$

P₄

18	19	25	26	34	36
1	2	3	4	5	6

$$M_4 = 25$$

M	27	32	33	35
1	2	3	n	5

$$M_5 = 32 \quad (\text{odd})$$

Step-III

make a list of all medians

14	9	7	25	32
1	2	3	4	5

sort the medians

9	7	14	25	32
1	2	3	4	5

Find the median = 14

$$M = 14$$

Step-IV

in the above algorithm $M = 14$

$$L = \{ s_i \in S; s_i < M \}$$

$$E = \{ s_i \in S; s_i = M \}$$

$$U = \{ s_i \in S; s_i > M \}$$

make a list

$$L = \boxed{3 | 8 | 1 | 4 | 9 | 10 | 5 | 13 | 7 | 2 | 12} \\ L$$

$$U = \boxed{16 | 20 | 3 | 22 | 33 | 19 | 18 | 25 | 26 | 34 | 36 | 27 | 32 | 33 | 35} \\ U$$

$$E = \boxed{14 | 14 | 14} \\ E$$

Step-V

The elements of $L = 11$

The elements of $U = 15$

The elements of $E = 3$

Computation

case 1 - if $|L| \geq k$ then select $\{L\}$

case 2 - if $|L| + |E| \geq k$ then return M

case 3 - else
select $(U, k - |L| + |E|)$

count no of elements in each case

(i) if $|L| \geq k$

(ii) if $|L| + |E| \geq k$

(iii) select $(U, k - (|L| + |E|))$

case-1 $|l| > 21$ → false

case-2 $(|L_1| + |L_2|) > k \Rightarrow n+3 > 21$ false

case-3 is exceeded, so element count is greater than n .
select $(n, k - (|L_1| + |L_2|))$

$$\text{now } K = 21 - (11+3) \\ = 21 - 14 = 7$$

$$\boxed{K=7}$$

step-6 consider only U array i.e.
here no of elements = 15

$$n = 0.5^{220}, n = 15 \\ P = (15)^{1-0.5^{220}}$$

$$P = 3$$

no of elements are assigned.

$$\text{no of procs} = (15)^{0.5^{220}} \approx 5$$

Three processes are assigned 5
elements each

P_1
16 20 34 22 33

P_2
19 18 25 26 34

P_3
36 27 32 33 35

find out the median of each
processor

for processor P_1 , median = 21

16 20 22 31 33

$$m = 22$$

for processors P_2

18	19	25	26	34
----	----	----	----	----

$$M_2 = 25$$

for processors P_3

27	32	33	35	36
----	----	----	----	----

$$M_3 = 33$$

Step - VII

make a list of all medians

22	28	33
\uparrow M		

$$\text{median} = \left(\frac{n+1}{2} \right)$$

$$= \frac{3+1}{2} = \frac{4}{2} = 2$$

$$M = 25$$

Step - VIII

~~Comparison~~

calculate $L = [16 | 20 | 22 | 19 | 18]$ $s_i < M$

5 element

25	
----	--

$$s_i = M$$

one element $P_2 = 1$ element

calculate $U = [31 | 33 | 26 | 34 | 36 | 27 | 32 | 39 | 35]$

$n = 9$ element

Step - IX ($80 - k = 7$)

case-1 $|L| + |s_i| \Rightarrow 5 \geq 7 - \text{false}$

case-2 $|L| + |s_i| \geq k \Rightarrow 5 + 1 \geq 7 - \text{false}$

case-3 is executed.

$$(n, k - (|L| + |s_i|))$$

new $k = 7 - (|L| + |s_i|)$

process will continue till the finding of the concerned case.