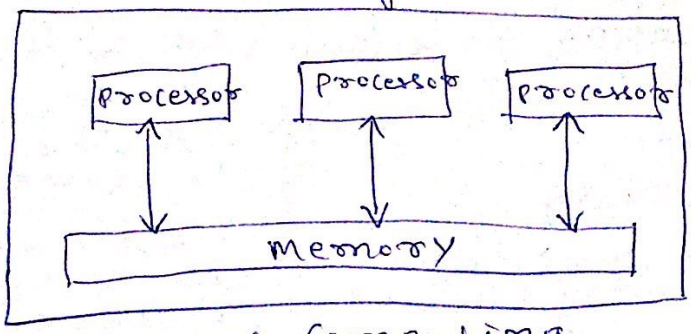
parallel and distributed computing

parallel computing.

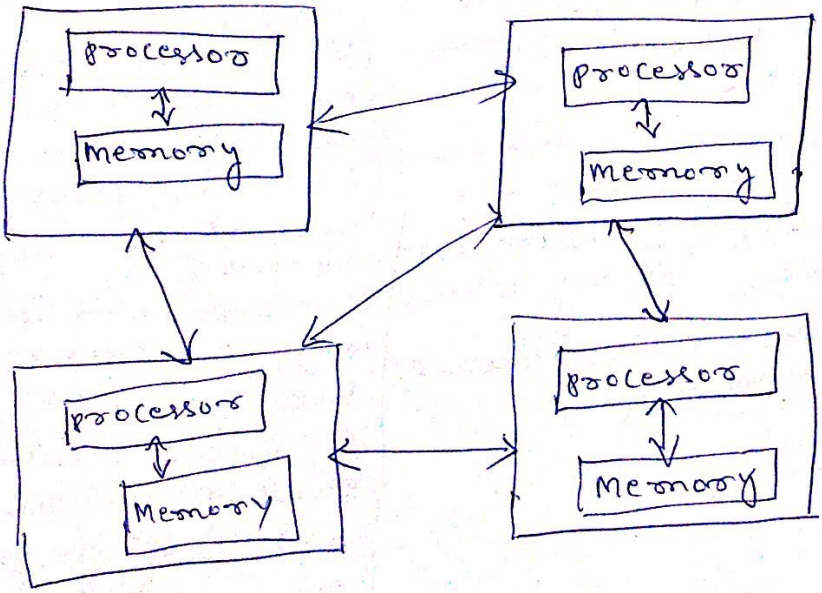
A parallel computing system consists of multiple processors that communicate with each other using a shared memory



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	VS	Distributed computing
i- Type of computation in which many calculations or the execution of process are carried out simultaneously		A system whose components are located on different networked computers which communicate and coordinate their action by passing messages to one another
ii- occurs in a single computer		ii- Involves multiple computers
iii- multiple processor executes tasks at the same time		iii- multiple computers perform tasks at the same time
iv- computer can have shared memory		iv- each computer has its own local memory.
v- processors communicate each other using a bus		v- computers communicate with each other via the network
vi- increase performance of the system		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 — As 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 reliability than parall system.

→ Data sharing — Data sharing provided by distributed systems is similar to data sharing provided by distributed databases.

→ Resources 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 systems.

→ 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 computers has its own memory

Usages ——— 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 processor. therefore parallel computing provides reliability.

parallel computing and distributed computing are two types of computation

* The main differences 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 applied to work is working at the same time.

Advantages —

→ it saves time and money as many resources working together ~~as m~~ 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.

→ Real world 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 can be organized only and only using parallel computings approach.

→ Ensures the effective utilization of the resources. The hardware is used effectively where as 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.

$$y = (4 \times 5) + (1 \times 6) + (5 \times 3)$$

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

step 1: $y = 20 + (1 \times 6) + (5 \times 3)$

step 2: $y = 20 + 6 + (5 \times 3)$

step 3: $y = 20 + 6 + 15$

step 4: $y = 41$

but in parallel computing scenario with three processors or computers

the steps look something like:
→ step 1: y = 20 + 6 + 15
→ step 2: y = 41
break the task down into pieces and execute those pieces simultaneously

MODELS OF COMPUTATION

concurrent processing

it can devide 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 parallesims

it is a process of processing several set of instructions simultaneously.
→ it reduces the total computational time
→ parallesims 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-

(i) Sequential Computer

(ii) Parallel computers.

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 devised into sub-problems and are executed in parallel to get individual outputs. 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 models of computation in a parallel computer.

Models of Computation

both sequential and parallel computers operate on a set of Instructions called algorithms.

→ 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 catagories.

I → single Instruction stream, single data stream (SISD) computers

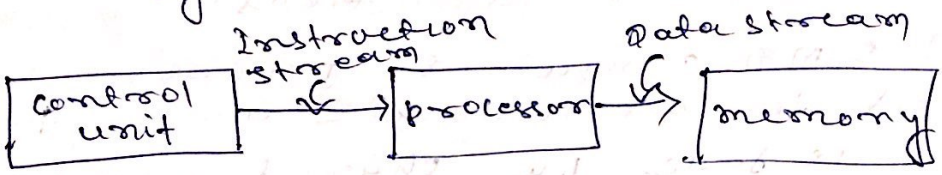
II → single Instruction stream, multiple Data stream (SIMD) computers

III → Multiple Instruction stream, ~~multiple~~ single data stream (MISD) computers

IV → Multiple Instruction stream, multiple data stream (MIMD) computers.

SISD computers —

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

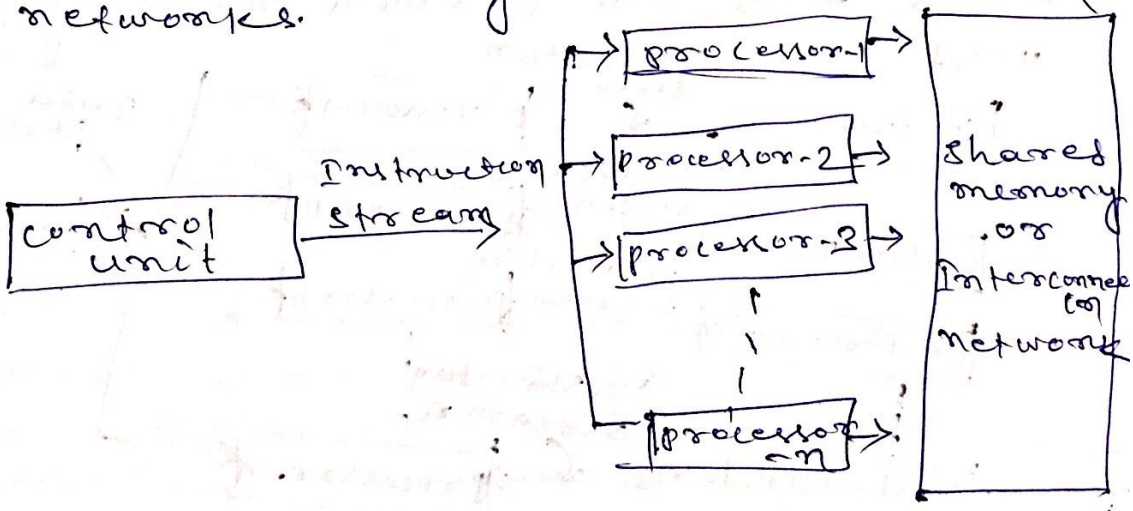


In this type of computers, the processor receives a single Instructions stream from the control unit and operates on single stream of data from the memory unit.

→ During computation, at each steap, 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 ~~receive~~ 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.

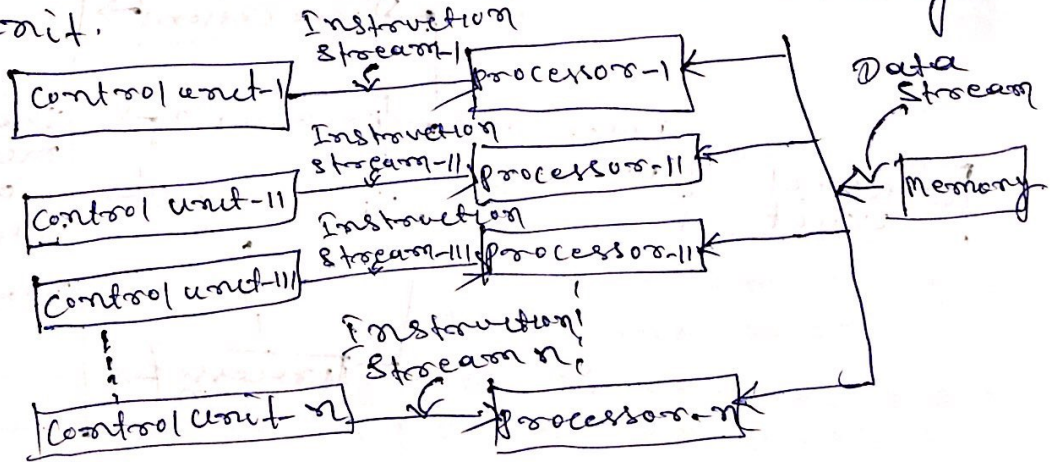
→ each of the processing units has its own local memory unit to store both data and instructions.

→ In SIMD computers, processors need to communicate among themselves. 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. Instructions from control unit decides which processor will be active (execute instructions) or inactive (wait for the next instruction).

MISD Computers

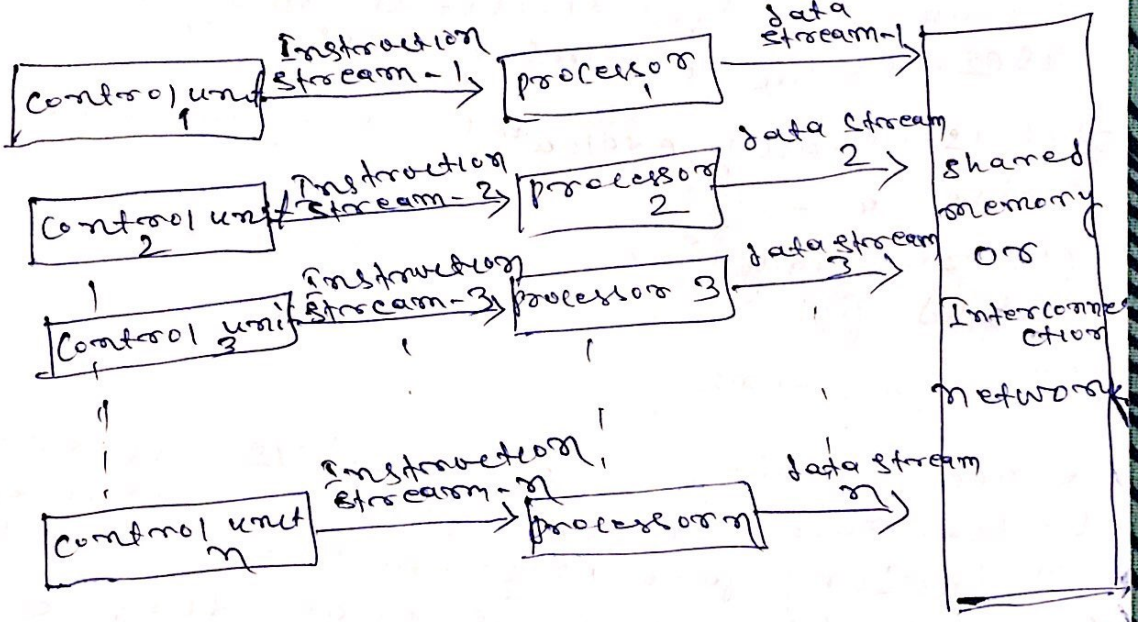
MISD computers contain multiple control units, multiple processing units and one common memory unit.



→ 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 networks.



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 operates on different sets of data.

* An MIMD computer that shares a common memory is known as multiprocessors while those that use an Interconnection network is known as multi computers.

multicomputer – 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 good an algorithm is, that is how fast, how expensive to run, and how efficient it is in its use of the 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 there for its running time.

* → it is desined as the time taken by the algorithm to solve a problem on a parallel computer. that is the time esapsed 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 is equal to the time elapsed between 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 theoritical analysis of the time it will require to solve the computational 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 arithmatic or logic operation performed on a piece of information with in a processor.

* routing steps — the piece of information travell from one processor to another via shared memory or through the communication network.

Ex — we want to compute the product of two $n \times n$ matrices. since the resulting matrix has n^2 entries. at least this may steps are needed by any matrix multiplicate 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 algorith is the number of processors it requires to solve a problem.

→ it costs money to purchase, ~~maintain~~ 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.

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

Ex 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

of n.

Cost

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

cost = parallel running time × number of processors used.

Efficiency = worst case running time of fastest known sequential algorithm for problem

---

cost of parallel algorithm.

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

Expressing parallel Algorithms

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

→ if we consider the algorithm is written in structured programming languages such as pascal.

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

the meaning of these statements assumes to be known.

a ← 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 expressions each taking one of the values true or false, then

(i) (a and b) is true if bot a and b are true; otherwise (a and b) is false.

(ii) (a or b) is true if at least one of a and b is true; otherwise (a or b) is false

(iii) (a xor b) is true if exactly one of a and b is true; otherwise (a xor b) is false; and

(iv) (not a) is true if a is false; otherwise (not a) is false;

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. □
```

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

```
for i = j to K do in parallel
{ The operations to be performed by Pi
  are stated here }
  end for □
```

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

for i = r, s, ..., t do in parallel
{
The 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
{The operations to be performed by P_i are stated here}
end for □

Where S is a given set of Integers

comments in algorithms are surrounded with curly brackets { },

MATRIX-MATRIX MULTIPLICATION USING PARALLEL PROCESSORS

Let each processing elements PEij represent two elements aij, bij.

- There are only n processing elements containing a pair of scalars suitable for multiplication.
- devise matrices A and B so that every processor has a pair of scalar that needed to be multiplied.
- The element A will move in leftward rotation and the elements of B moves in upward rotation

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

Step-1

Divide the matrix into different parts
- The second computes all products $a_{ik} \times b_{kj}$ and accumulates sums when the phase 1 is completed.

Algorithm

```
Procedure MATRIXMULT
begin
for k=1 to n-1 step 1 do
 begin
for all Pij, where i and j ranges from
1 to n do
if i is greater than k then
rotate aij in the east direction
end if
if j is greater than k then
rotate b in south direction
end if
end
for all Pij where i and j lies between
1 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
Pij where i and j ranges from 1 to n
rotate a in the east
rotate b in the south
    C = C + a x b
    end
```

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

The matrix elements are stored in two dimensional mesh

$A = \left(\begin{array}{c|c} a_{11} & a_{12} \\ 1 & 2 \\ \hline 3 & 4 \\ a_{21} & a_{22} \end{array}\right)$ $B = \left(\begin{array}{c|c} b_{11} & b_{12} \\ 5 & 6 \\ \hline b_{21} & b_{22} \\ 7 & 8 \end{array}\right)$

$$\text{mesh} = \begin{pmatrix} P_{11} & P_{12} \\ 1/5 & 2/6 \\ P_{21} & P_{22} \\ 3/7 & 4/8 \end{pmatrix}$$

let the number of processor be 4
that is $n^2 = 4 \Rightarrow n = 2$

no of processor be 4. that is

$P_{(1,1)}$, $P_{(1,2)}$ $P_{(2,1)}$, $P_{(2,2)}$ 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 is greater than k and rotate a_{12}, in the left (east) direction

→ apply and a_{21}, and a_{22} changes its place from leftwards (row wise)

→ and b_{12} and b_{22} also changes its place from bottom to up (south direction)

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

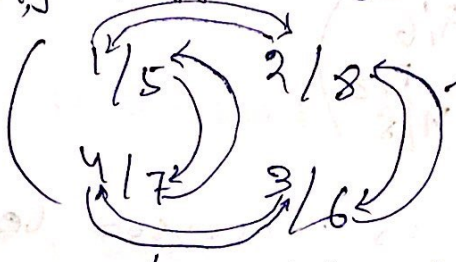
upward (south direction)

leftward (row wise)

The processors $P_{i,j}$ ranges from 1 to 2 will compute the product of a_{ij} and b_{ij} respectively

$$C_{ij} = \begin{pmatrix} 1\times5 & 2\times8 \\ 4\times7 & 3\times6 \end{pmatrix} = \begin{pmatrix} 5 & 16 \\ 28 & 18 \end{pmatrix}$$

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

mesh = 

left shift (row wise)

mesh

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

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

Now the processors $P_{i,j}$ compute the product of a_{ij} and b_{ij} respectively with old value of C_{ij}

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

$$= \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} = \text{Resultant matrix}$$

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

Ex-2

$$A = \begin{bmatrix} 2 & 1 & 5 & 3 \\ 6 & 7 & 1 & 6 \\ 9 & 2 & 4 & 4 \\ 3 & 6 & 7 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 4 & 5 & 6 & 5 \\ 1 & 9 & 8 & -8 \\ 4 & 0 & -8 & 5 \end{bmatrix}$$

if we consider a sequential algorithm the computation will perform serially element by element and stores the result in memory and then as individual elements

$$\begin{bmatrix} 2\times6 + 1\times4 + 5\times1 + 3\times4 \end{bmatrix}.$$

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

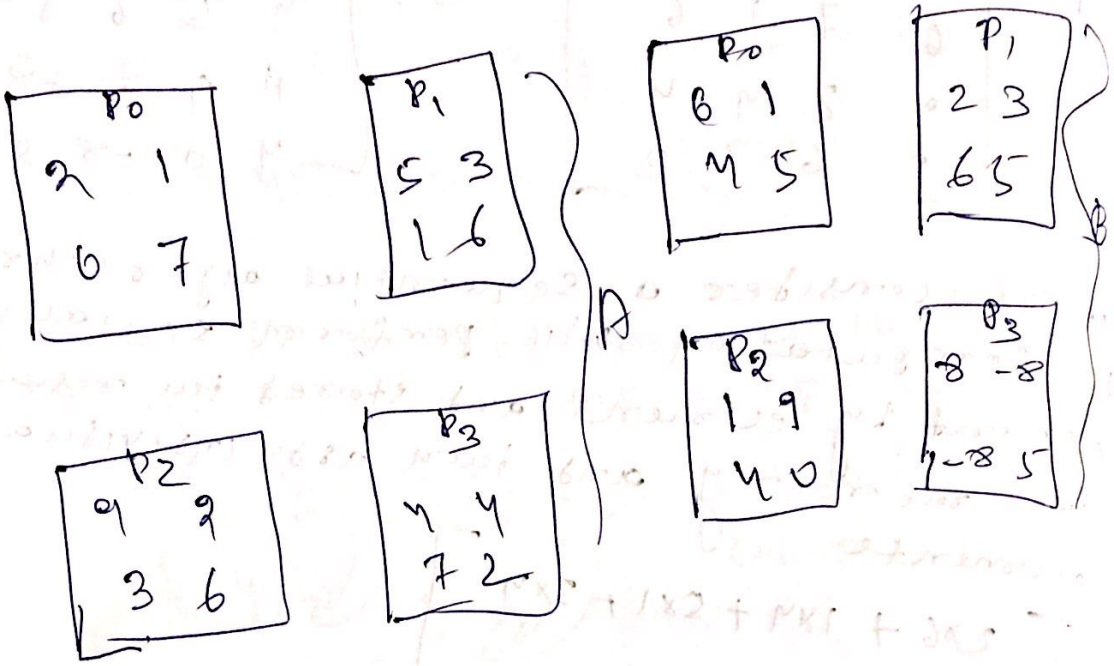
$= 12+4+5+12 = 33$

but if we consider in matrix-matrix multiplication in parallel processors

step-1

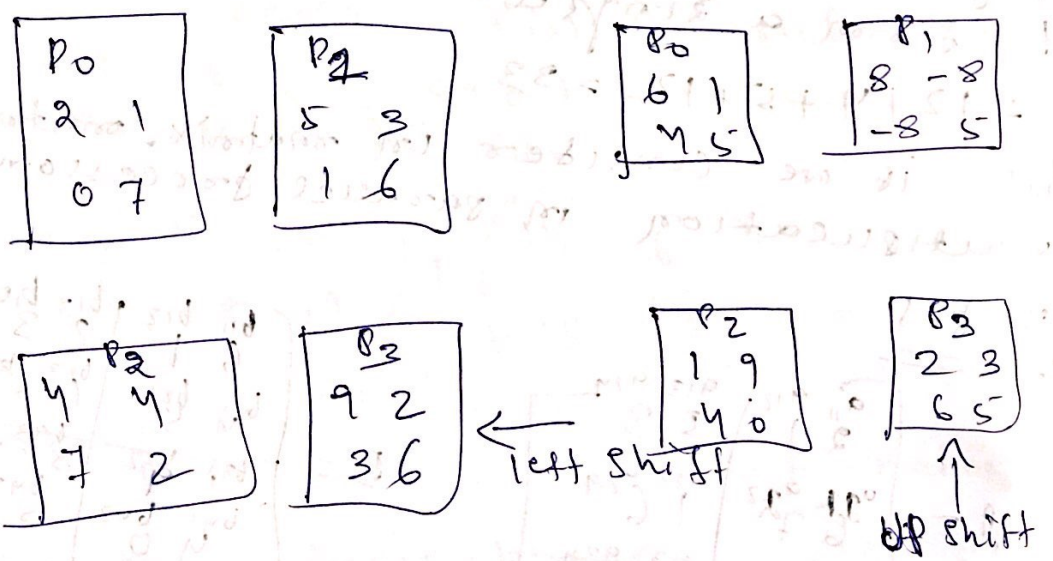
$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ 2 & 1 & 5 & 3 \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 6 & 7 & 1 & 6 \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ 9 & 2 & 4 & 4 \\ a_{41} & a_{42} & a_{43} & a_{44} \\ 3 & 6 & 7 & 2 \end{array}\right] \qquad B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ 6 & 1 & 2 & 3 \\ b_{21} & b_{22} & b_{23} & b_{24} \\ 4 & 5 & 6 & 5 \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ 1 & 9 & 8 & -8 \\ b_{41} & b_{42} & b_{43} & b_{44} \\ 4 & 0 & -8 & 5 \end{array}\right]$$

assign processor to the both the devided matrix A and B
Suppose there are 4 processors



Suppose there are 4 processors
$n^2 = 4 \Rightarrow n = \sqrt{4} = 2$ steps are required for computation

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



Step - II

Apply Computation → multiply processor P_0 of matrix A to P_0 of matrix B resultant matrix

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

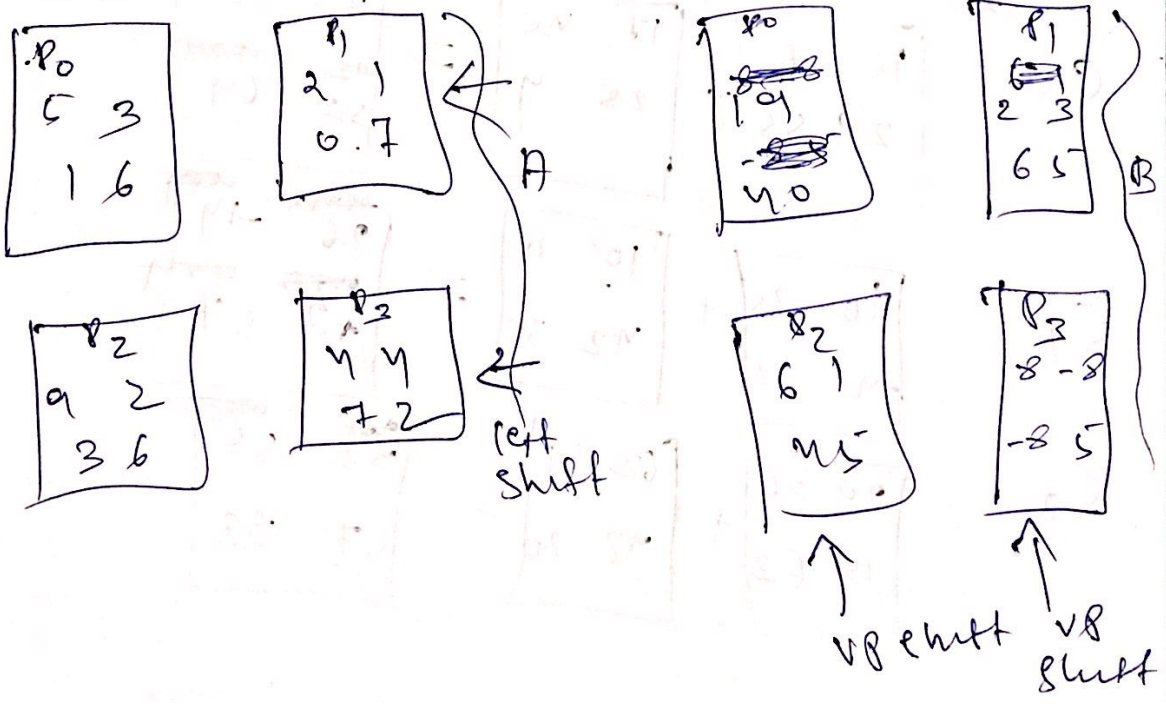
Similarly $C_1 = \begin{bmatrix} P_1 \\ 5 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} P_1 \\ 8 & -8 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 16 & -24 \\ -40 & 22 \end{bmatrix}$

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

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

Step - III

Again apply leftshift and upshift to all the processors of matrix A and B



Array Computation

$C_0 = \begin{bmatrix} 5 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 9 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix}$

$C_1 = \begin{bmatrix} 2 & 1 \\ 0 & 7 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix}$

$C_2 = \begin{bmatrix} 9 & 2 \\ 3 & 6 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 62 & 19 \\ 42 & 20 \end{bmatrix}$

$C_3 = \begin{bmatrix} 4 & 4 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 8 & -8 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 0 & -12 \\ 40 & 16 \end{bmatrix}$

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.

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

$C_1 = \begin{bmatrix} 16 & -25 \\ -40 & 22 \end{bmatrix} + \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix} = \begin{bmatrix} 26 & -14 \\ +2 & 57 \end{bmatrix}$

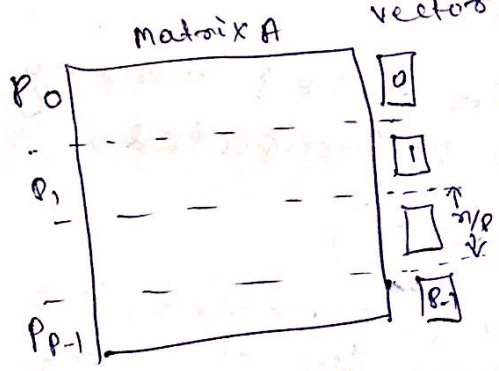
$C_2 = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix} + \begin{bmatrix} 62 & 19 \\ 42 & 20 \end{bmatrix} = \begin{bmatrix} 82 & 55 \\ 57 & 83 \end{bmatrix}$

$C_3 = \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix} + \begin{bmatrix} 0 & -12 \\ 40 & -46 \end{bmatrix} = \begin{bmatrix} 30 & 25 \\ 86 & -7 \end{bmatrix}$

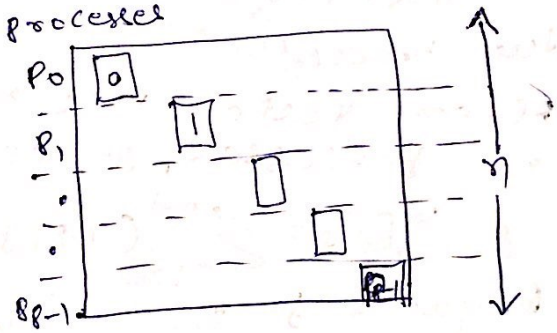
$C = \begin{bmatrix} C_0 & C_1 \\ C_2 & C_3 \end{bmatrix} = \begin{bmatrix} 38 & 52 & 26 & -14 \\ 53 & 64 & +2 & 57 \\ 42 & 55 & 30 & 25 \\ 57 & 83 & 86 & -7 \end{bmatrix}$

MATRIX VECTOR MULTIPLICTION

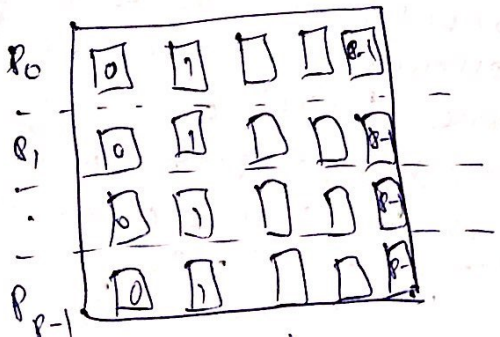
multiplication of an $n \times n$ matrix with an $n \times 1$ vector using row wise block 1-D partitioning for one row-per-process case, $p = n$



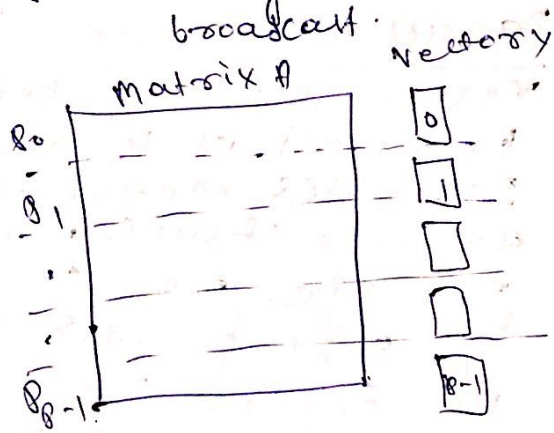
(a) Initial partitioning of the matrix and the starting vector X



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



(c) Entire vector distribution to each process after the 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. The $n \times 1$ vector x is distributed such that each process owns one of its elements. The initial distribution of the matrix and the vector for rowwise block 1-D partitioning is shown in fig (a)

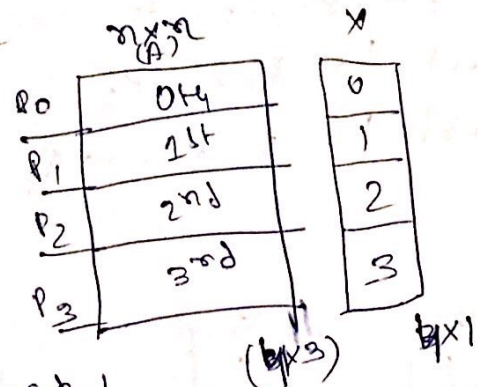
- Process P_i initially owns $x[i]$ and $A[i,0], A[i,1], \ldots A[i,n-1]$ and is responsible for computing $y[i]$
- Vector x is multiplied with each row of the matrix. So every process needs the entire vector. Since each process starts with only one element of x, an all-to-all broadcast is required to 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 (b) 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, the all-to-all broadcast of the vector elements among n processes requires time $\Theta(n)$ on 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 $\Theta(n^2)$

P2X



Step-I

no of rows 4, 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 1stth element of Vector X

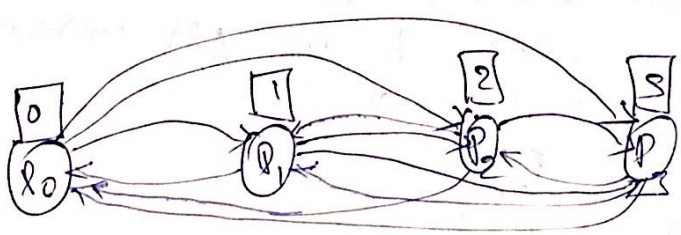
in this manner 2nd row of matrix A is given to P_2 process and is assigned to 2ndth element of vector X

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

Step-II

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 2ndth element of X and P_3 is holding only 3rd element of X

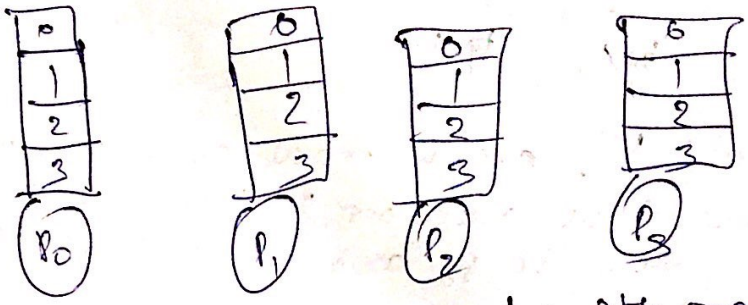
so broadcast is necessary (all to all broadcast)



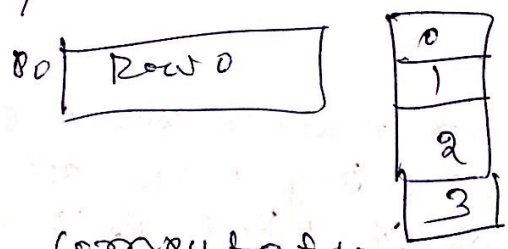
broadcast P_0 element of X to P_1, P_2 and P_3

P_1 current is broadcast to P_0, P_2, P_3.
P_2 current is broadcast to P_1 P_0 P_3
P_3 current is broadcast to P_0 P_1 and P_2.

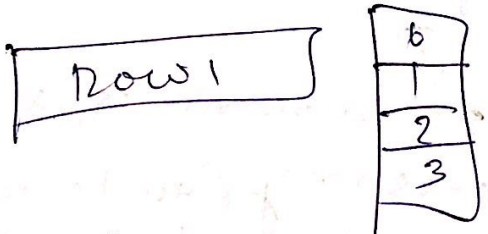
Step-IV



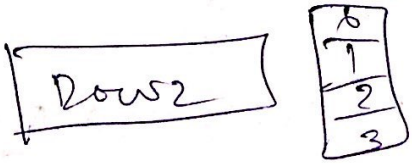
So P_0 process is assigned to 0th row of matrix X and now every process has a unique vector (complete vector)
So P_0 is holding 1st row as well as complete vector. similiarly all P_1, P_2 and P_3 also contains 1st complete vector individually.
Now multiply process P_0's row with vector / row 0 with vector.



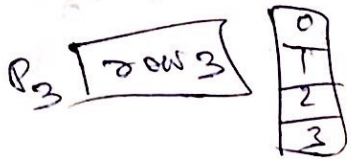
after computation store the result in 0th index of R



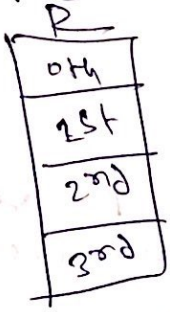
after computation row1 with vector store the result in 1st index of Resultant vector R



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



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



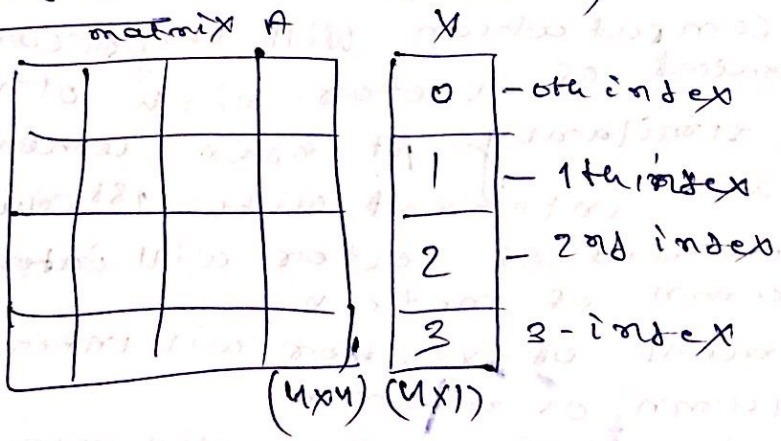
Resultant vector

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

it consists of a n×n matrix multiplied with vector (n×1)

→ it consists of (matrix) rows and columns

Step-1



→ one element per process for computation purpose

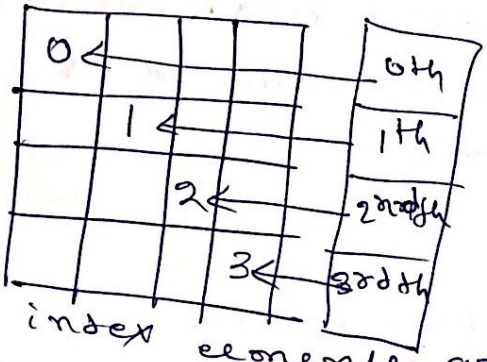
→ Total no of processes $p = n^2$ elements.

For each element one process is assigned

Step-11

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

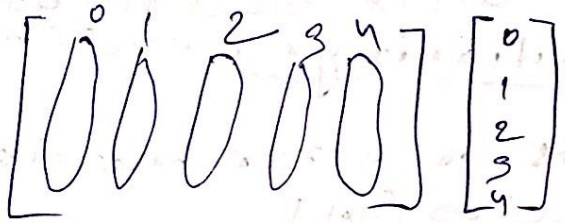
Put 0th index, 1th index, 2th index and 3rdth index of vector (x) bellow or given picture



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

Step-III

apply one to all communication because



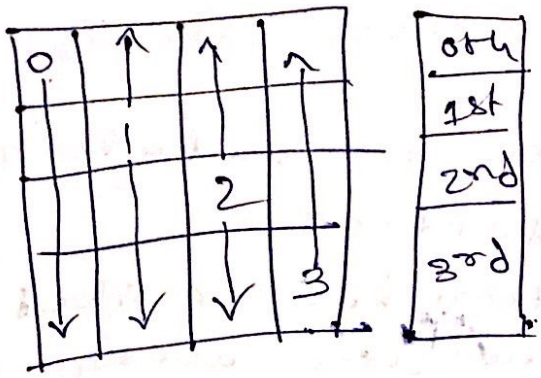
in the above picture 0th index element of vector is multiplyed with 0th column of the matrix. it will not interact to other. so computation will only occurs 0th index element of vector with 0th column and similarly 1st index element of vector will interact with 1st column

→ 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. for the computation purposes.

Step-IV

apply computation because every cell contains element and each cell assigns one process each in ~~row~~ column wise. So in the given picture above ~~row~~ 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 multiply with 1st column.

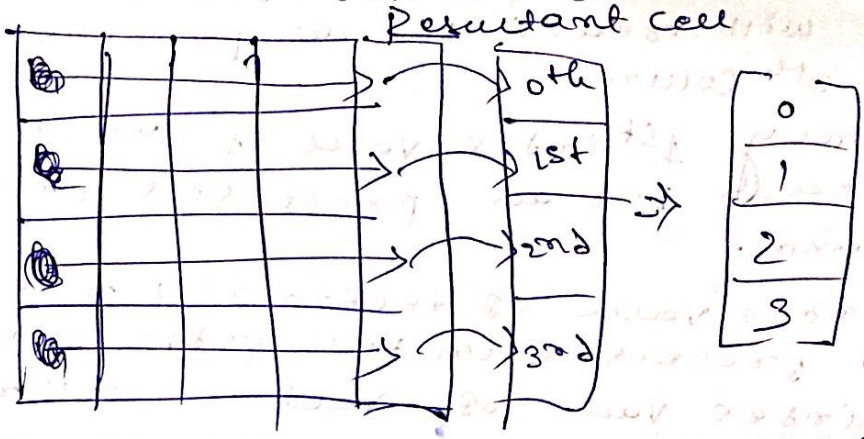
computation occurs 2nd index element of vector with 2nd column elements

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.



Accumulate every cell element row wise and adding the elements row wise cells element and stored in to the result and vectors 0th index, 1th index(~~second~~ 1st row elements), 2nd th index(2nd row cell elements), and 3rd index(3rd row cell elements)

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.

Database Query processing in parall computers

The processing of a SQL statement is always performed by a single server process

→ With the parallel query features, multiple processes can work together simultaneously to process a single SQL statement

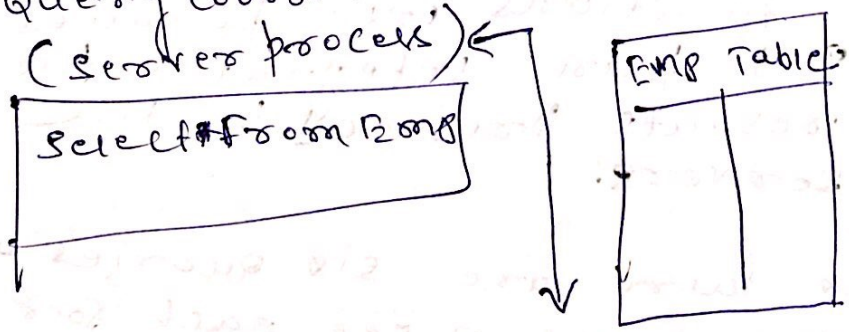
This capability is called parallel query processing

- by devising 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.

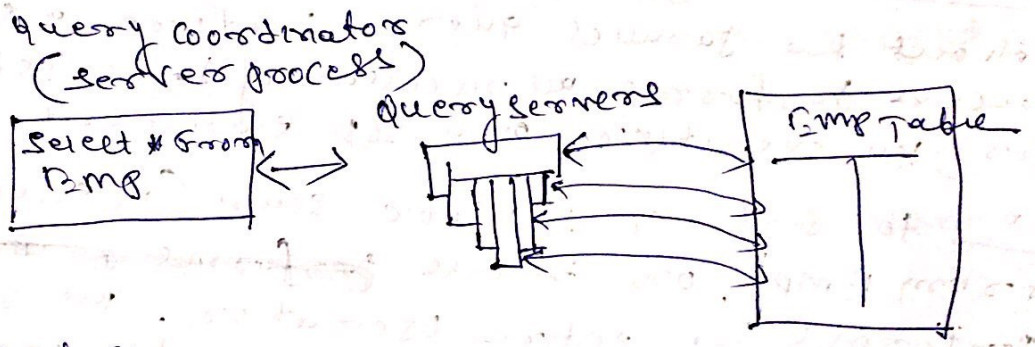
Ex to perform full table scan (Ex select * from EMP), one process performs the entire operation. The following figure illustrates a server process performing a full table scan.



(full table scan without the parallel query feature)

The parallel query feature allows certain operations (for example, full table scans or sorts) to be performed in parallel by multiple query server processes, dispatches 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 server processes 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.

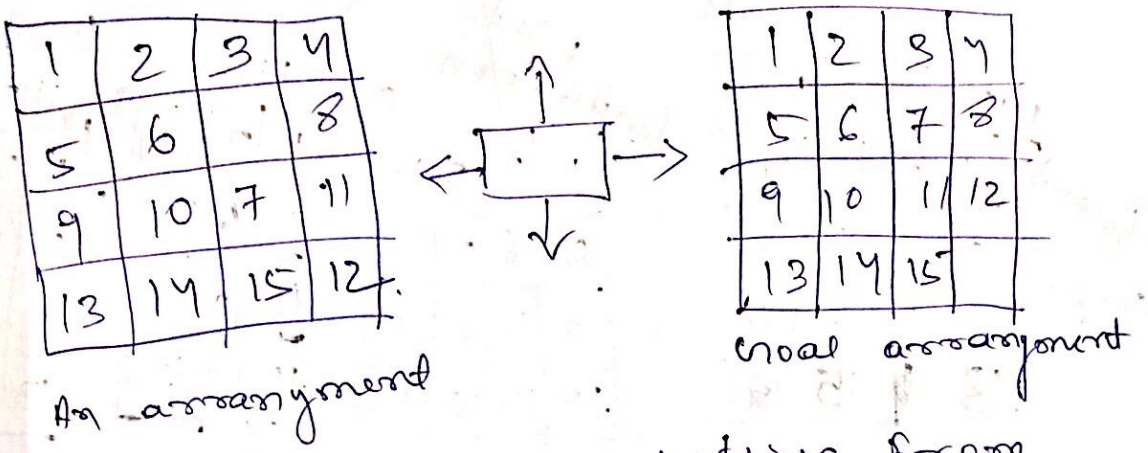
— The query coordinator can breakdown execution functions into parallel pieces and then integrates the partial results produced by the query servers.

Suppose Ex there are six Queries and each query take 3 sec each for execution.

So total time it take to execute
= 6 x 3 = 18 seconds for all Queries in sequentially

but is the queries and sub devided into different parts and each sub parts are assigned to different processors. So execution of all the queries will be completed with in the 3 seconds

15 PUZZLE PROBLEM



An arrangement

Goal arrangement

we will move the blank plate/tile from up, down, left and right position after that 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 minimum error.

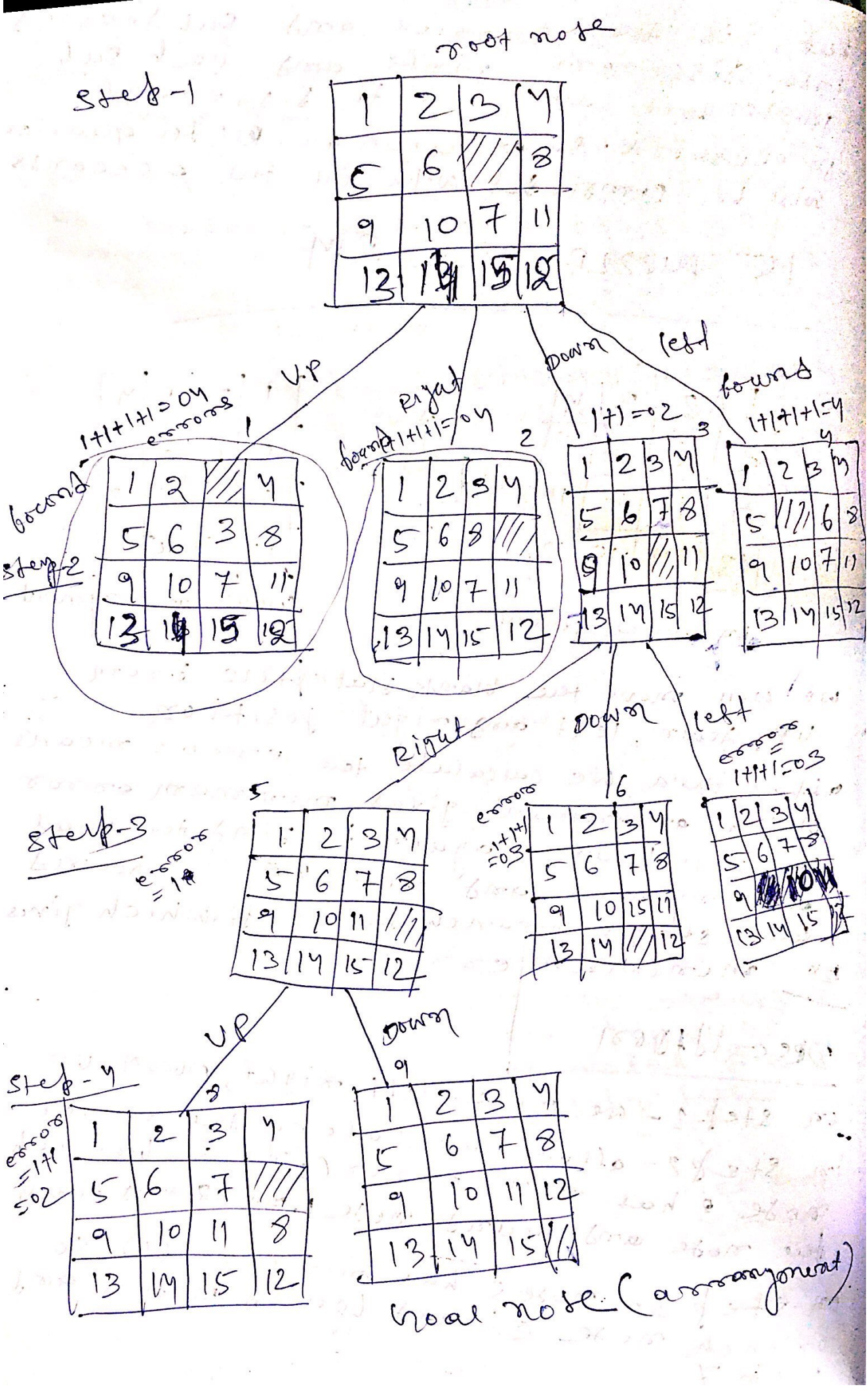
Ex

Description

in step-1 - we move left, right, down, up

in step 2 - after calculating the error node 3 has min error (2). So branch the node and bound node 1, node 2 and node 4

in step 3 - node 5 has min error (1) So branch node 5 and bound node 6 and node 7



in step-4 node 9 is the resultant goal.

parallel Discreate event simulation

- A simulation model can be viewed as a representation of the physical system under simulation.

Discreate event simulation (DES) are characterized by discreate-state models (or 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 discreate state model, or discreate simulation, the state of the simulation is instaneous and occurs at discreate point in simulation time (the discreate 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 ~~simulated~~
- is maintained in a global event list
- These events are to be scheduled by an event scheduler in a non-decreasing time stamp 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 processors

Principles of parallel simulation

In PDES, a model generally comprises N logical processors (LPs) LP0, LP1 ---, LPN-1 which interact among themselves by sending timestamped event messages. A link exists from LPi to LPj if LPi may send messages to LPj.

- each message is a tuple (Ek, Tk) consisting of an event Ek and its associated time stamp Tk.
- each LPi is associated with a local clock, or local virtual time (LVT), clocki is automatically advanced to Tk.

LET E1 and E2 denote two events with timestamps T1 and T2 respectively → Assume that T1 < T2. If E2 depends on E1, then there is a causality constraint over the order of execution of E1 and E2. For instance, if E1 causes a change of state variable which will be referenced by E2, then E1 must be executed before E2 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 formed as a local causality constraint. it is important 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 error.

MODULE-II

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.

Therefore any two elements, a_i and a_j any one of the 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 P1,j do
    C[j] := 0;
for each process Pi,j do
if ( A[i] < A[j] or A[i] = A[j] and i<j)
```

```
else
    C[j] := 0;
for each process P1,j do
    A[C[j]] := A[j];
end ENUM_Sorting
```

Ex — A = { 2, 5, 0, 4 }
1 2 3 4

partition

2	[2,2] P(1,1) ⓪	[2,5] P(1,2) ⓪	[2,0] P(1,3) ①	[2,4] P(1,4) ⓪	— 0+0+1+0=1
5	[5,2] P(2,1) ①	[5,5] P(2,2) ⓪	[5,0] P(2,3) ①	[5,4] P(2,4) ①	— 1+0+1+1=3
0	[0,2] P(3,1) ⓪	[0,5] P(3,2) ⓪	[0,0] P(3,3) ⓪	[0,4] P(3,4) ⓪	— 0+0+0+0=0
4	[4,2] P(4,1) ①	[4,5] P(4,2) ⓪	[4,0] P(4,3) ①	[4,4] P(4,4) ⓪	— 1+0+1+0=2

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

Ex — [2,2] P(1,1) in this case i=j place 0

[2,5] P(1,2) — i<j here is also place 0

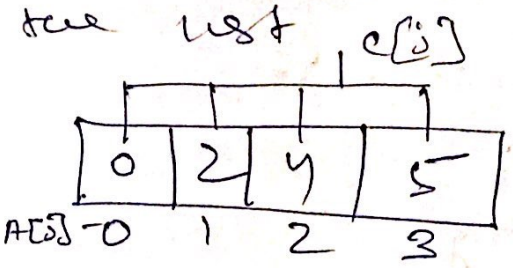
[2,0] P(1,3) — in this case i>j place 1

[2,4] P(1,4) — here in this case i<j place 0

Step-II

change the position
that is element 4 is placed in 2nd the position in the list
0 is placed 0th element in the list
2 is placed 1st position in the list

5 is placed in the 8th element in the list



A[i] = A[C[i]] means

A[0] = A[C[0]]

=> A[0] = A[0] = 0

A[1] = A[C[1]]

= 2

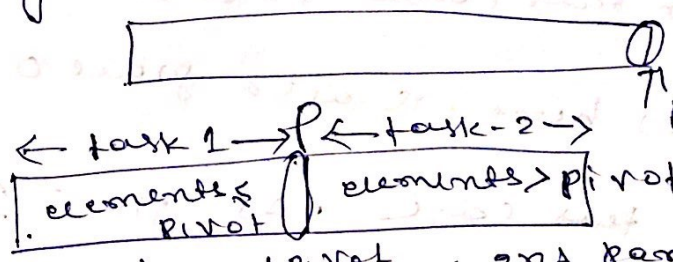
A[2] = A[C[2]] = 4

A[3] = A[C[3]] = 5

Parallel Quick Sort

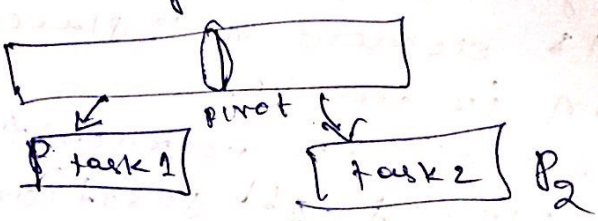
Step-1

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



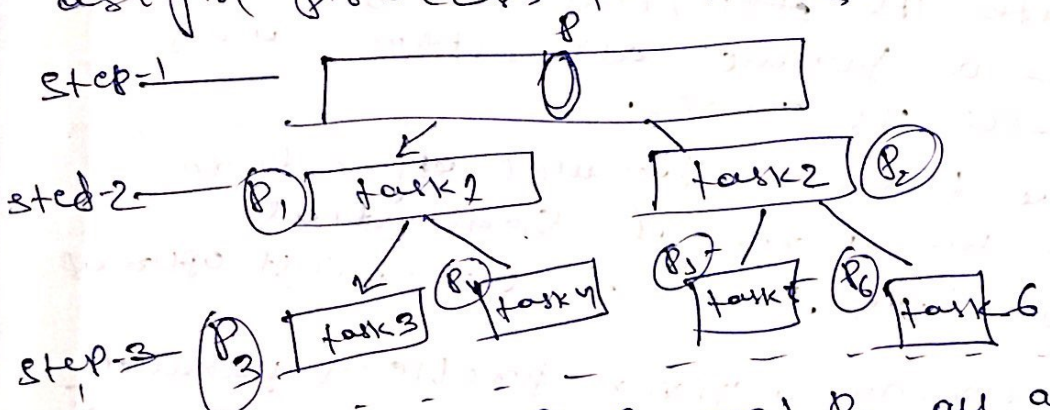
processor P is partitioned the array into two parts. after partitioned the two task are independent to each other.

assign the subtask for process P_1 and process P_2



So P_1 process will execute task1 and process P_2 will execute task-2 independly parallly

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



Processors P_3, P_4, P_5 and P_6 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 devides the task into sub tasks (P-single process devises the task in to task1 and task2 P_1 devises the subtask1 into task3 and task4

similarly P_2 also devides the task into task5 and task6

similarly process P_3, P_4, P_5 and P_6 devides 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.

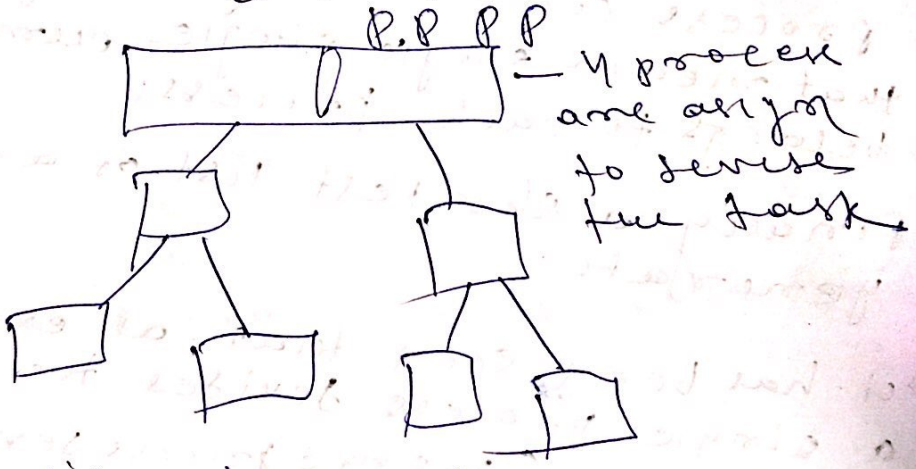
Cost optimal means if we take a best sequential algorithm and calculate its ~~cost~~ Time and we take a parallel algorithm and calculate its time.

if the time of parallel algorithm > than the Time of sequential algorithm. then it is not cost optimal

* for solving a given problem parallely if its take 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



4 process are assign to devide the task.

if partitioning step is performed in time $O(1)$, using $O(n)$ processes

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

* we will parallelize partitioning steps by using n process

Hyper Quick sort

Hyper Quick sort is an implementation of Quick sort on hypercube. Its 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

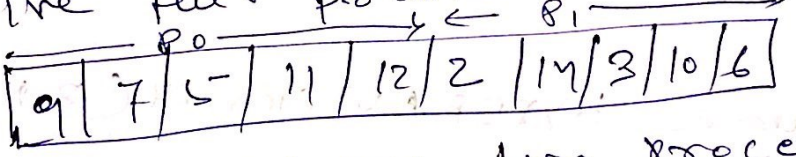
Algorithm

```
procedure HYPERQUICKSORT (B, N)
begin
    id := processes label;
for i := 1 to d do
begin
    x := pivot;
partition B into B1 and B2 such
that  B1 ≤ x < B2;
if ith bit is 0 then
begin
send B2 to process along the ith
communication Link
```

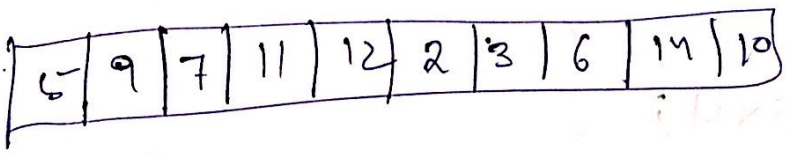
C'i = Subsequence received along the
ith communication link;
B'i = B1 ∪ C;
end if
else
send B1 to process along the ith
communication link;
C'i = Subsequence received along the
ith communication link;
B'i = B2 ∪ C;
end else
end for
Sort B using sequential sort;
end HYBRIDQUICKSORT;

Ex

Given a array of elements given
below. by using parallel processors
solve the problem



suppose there are two processors
P0 and P1. assign processor P0 to 1st
part and P1 to
suppose 6 is pivot element second part



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.
- if both the phases, processes exchange numbers with their adjacent numbers in the right.

Ex

1 2 3 4 5 6 7 8 unsorted

9 7 3 8 5 6 4 1 - phase 1 (odd)

7 9 3 8 5 6 1 4 - phase 2 (even)

7 3 9 5 8 1 6 4 - phase 3 (odd)

3 7 5 9 1 8 4 6 - phase 4 (even)

3 5 7 1 9 4 8 6 - phase 5 (odd)

~~3 5 7~~

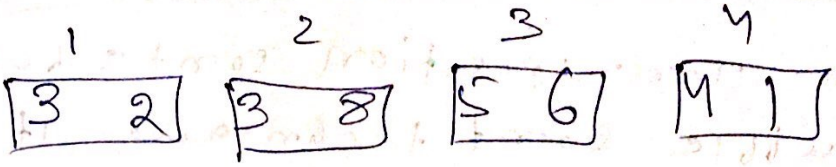
3 5 1 7 4 9 6 8 - phase 6 (even)

3 1 5 4 7 6 9 8 - phase 7 (odd)

1 3 4 5 6 7 8 9

sorted

Ex



suppose there are eight process are there and each process holds a single element.

— in this problem eight process are there and each holding one elements in the array.

P_1 - holding 1th element in the array
P_2 - " 2nd " " " "
P_3 - " 3rd th " " " "
P_4 - " 4th " " " "
.
.
P_8 - holding 8th " " " "

we will given the name odd/even by seeing its index label

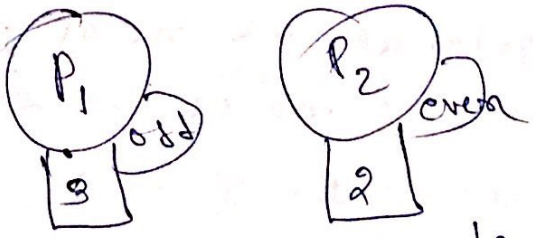
as Ex process P_1 if we consider
P_1 - index is odd - so we will given label odd

similarly P_2 - its 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 in odd face or even face

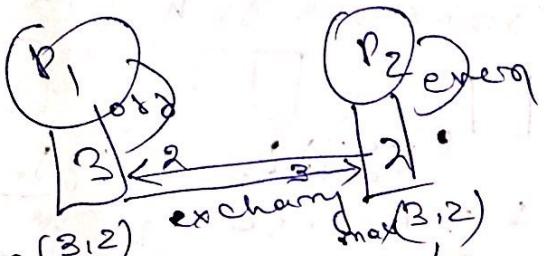


You have to check the left element of array and its label is odd so it is formed in odd fall [square is formed in odd fall]

Process P(1) - holding the 1st index value

calculating the even fall is same as that of above

- check the left element and if the index is even and label is even then square is formed in even fall.



then process P_1 and process P_2 will exchange their value [they will communicate to each other]

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

[2,3]

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

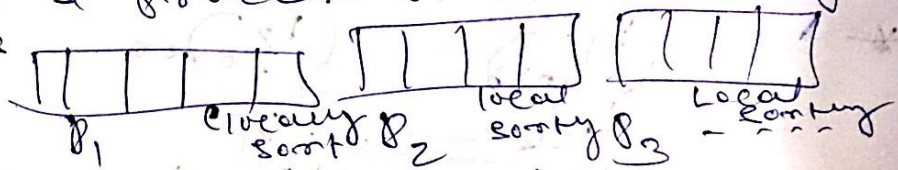
Cost optimality

if no of process < no of elements that means each process holds = (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

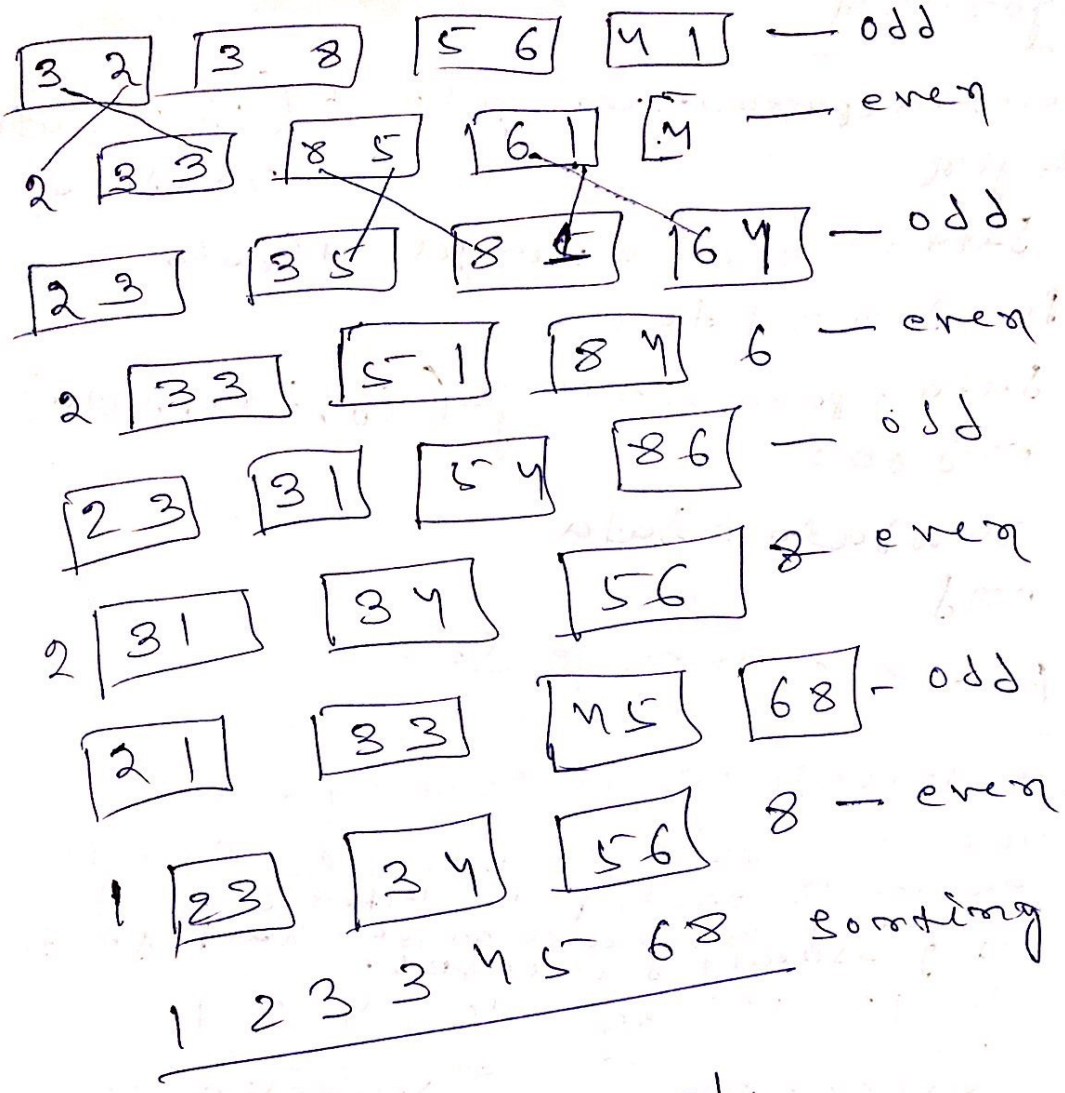
Time it take for local sorting

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

II :- communicating for exchanging of data - $\theta(n)$

III - comparision - $\theta(n)$

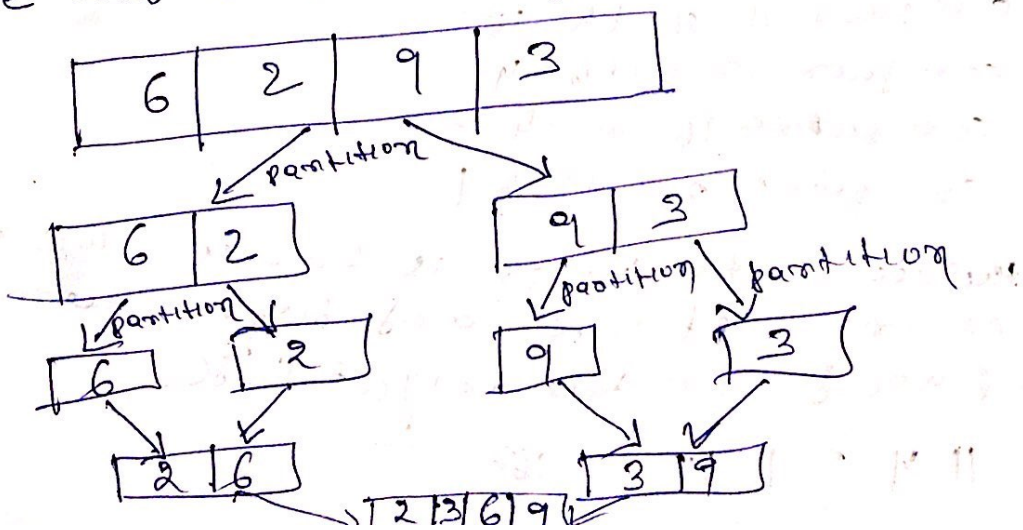
Total time complexity - $T_p = \theta(n/p \log n/p) + \theta(n) + \theta(n)$



parallel merge sort

Merge sort first divides the unsorted list into smallest possible sub-list compares it with the adjacent list, and merges it in a sorted order. it implements parallelism very nicely by following the divide and conquer algorithm.

Ex



Algorithm

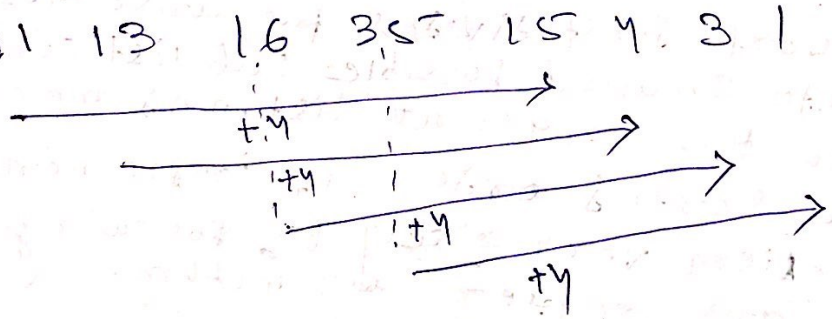
```
procedure parallelmergesort (id, n, data,
  begin                              newdata)
    data = Sequentialmergesort (data)
    for dim = 1 to n
    data = parallelmerge (id, dim, data)
    end for
    newdata = data
    end
```

Bitonic merge sort

It is procedure of sorting a bitonic sequence using bitonic sprite
a sequence is said to be bitonic if it
Ex grasually increases and after half position it grasually decreases.

11 13 16 35 15 4 3 1.

ascending order → : ← descending order



compair 11 with 15.
compare 13 with 4
Compare 16 with 3
Compare 35 with 1

the elements 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 (next next element)

11 4 3 1 | 15 13 16 35

+2 +2 +2 +2

3 1 11 4 15 13 16 35

compare 11 with 3

compare 4 with 1

in the part 2 compare 15 with 16

and compare 13 with 35

→ 3 1 11 4 15 13 16 35

→ Comparision with +1 element

means compare 3 with 1, 11 with 4

15 with 13, and 16 with 35

3 1 → 11 4 → 15 13 → 16 35 →

1 3 4 11 13 15 16 35

sorted

BX-2

3 7 4 8 6 2 1 5

3 → down arrow

7 ↓

4 ↑ UP arrow

8 ↓

6 ↑

2 ↑

1

5

3

7

8 ↓

4 ↑ mid position

2 ↑

6 ↑

5 ↑

1 ↑

3

4 ↓

8

7 ↓ mid position

5 ↑

6 ↓

2 ↑

1

→ 3 4 7 8 1 6 5 2 order

increasing decreasing

* V8 arrow - bigger element is replaced to smaller one
* down arrow - smaller element is replaced.

Description

an array arr[0...n-i] is bitonic if there exists an index i where $0 <= i <= n-1$ such that

$X0 <= X1 - \cdots <= Xi$ and $Xi >= X_{i+1} - \cdots >= X_{n-1}$

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-elements bitonic sequences from consecutive 2-elements sequence.

Consider 4-elements in sequence X0, X1, X2, X3. We sort X0 with X1 in ascending order and X2 and X3 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.

a) = min(a,b)
b) max(a,b)

a) max(a,b)
b) min(a,b)

Analysis of bitonic sort

To form a sorted sequence of ~~[illegible]~~ length n from two sorted sequences of length n/2, log(n) comparisons are required

→ $T(n) = \log(n) + T(n/2)$

The solution of this recurrence equation is

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

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

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

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

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

EREW MODEL

→ let X

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

EREW SORT:

procedure EREWSORT(S)

if $|S| \leq k$

then QUICKSORT(S)

else(1)

for i = 1 to k-1 do

PARALLELSELECT $(S, \lceil i|S|/k \rceil)$

{obtain m_i}

coy 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
BRFWSORT(Si)
end for

(6) for i=(k/2)+1 to k do in parallel
BRFWSORT(Si)
end for
end if □

B20

A sequence of integers
$S = \{s_1, s_2 \ldots s_n\}$ and an integer k are given and it is required to find the kth element

Algorithm

→ p no of processors are available
→ each processor will receive $[S]^x$ elements such that $(p = n^{1-x})$
n - no of elements
→ find median of each subsequence $[S_i/2]$ and place it in M.
→ Now find median of M. that is $[M/2]$
→ sequences divided into L, E, G where L - low, E = equal, G = Greater

$L = \{S_i \in S ; S_i < M\}$
$E = \{S_i \in S ; S_i = M\}$
$G = \{S_i \in S ; S_i > M\}$

3	14	16	20	8	31	1	4	9	12	22	33	2	5	7	10	13	14	19	18	25	26	34	36	11	27	32	33	35

BY

3	14	16	20	8	31	22	33	1	4	9	12	2	5	7	10	13	14	19	18	25	26	34	36	11	27	32	33	35

33

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

$k^{th} = 21^{st}$ $p = 5$ $n = 29$

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

$p = n^{1-n}$

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

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

$\Rightarrow n = 0.5220$ ——— (1)

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

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

So every processor will assign 6 elements each.

P_1

3	14	16	20	8	31

P_2

1	4	9	12	22	33

P_3

2	5	7	10	13	14

P_4

19	18	25	26	34	36

P_5

11	27	32	33	35

5 element

step-1)

Sorting each processor elements and calculate the median value of each processor

after sorting

P_1

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

Median is $M_1 = 6/2 = 3^{rd}$ positions element

$M_1 = 14$

P_2

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

$M_2 = 6/2 = 3^{rd}$ element

$M_2 = 9$

P_3

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

$M_3 = 7$

P_4

18	19	25	26	34	36

$M_4 = 25$

P_5

11	27	32	33	35
1	2	3	4	5

$M_5 = 32$ $(n+1/2)$

Step-III

make a list of all medians

14	9	7	25	32

sort the median

9	7	14	25	32
1	2	3	4	5

Find the median = 14

$M = 14$

Step-IV

in the above algorithm M = 14

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

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

$G = \{ s_i \in S ; s_i > m \}$

make a list

L = | 3 | 8 | 1 | 4 | 9 | 10 | 5 | 13 | 7 | 2 | 12 |

L

G = | 16 | 20 | 31 | 22 | 33 | 19 | 18 | 25 | 26 | 34 | 36 | 27 | 32 | 53 | 35 |

G

E = | 14 | 14 | 14 |

Step-V

The elements of L = 11

The elements of G = 15

The elements of E = 3

Comparison

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

case 2 - if $|L| + |E| \geq K$ then Return M

case 3 else

select $(G, K - |L| + |E|)$

count no of elements in each case

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

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

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

case-1 $|L| \geq 21$ — false

case-2 $|L_1| + |L_2| \geq k \Rightarrow 11+3 \geq 21$ false

case-3 is executed. so element is present in S.

select $(S, k-(|L_1|+|L_2|))$

now $k = 21-(11+3)$

$= 21-14 = 7$

$\boxed{k=7}$

step-VI

consider only S array list here in S no of element = 15

$n = 0.5^{220}$, $n = 15$

$p = (15)^{1-0.5^{220}}$

$p = 3$

no of elements are assigned per process $= (15)^{0.5^{220}} \simeq 5$

Three processors are assign 5 elements each

P_1

16	20	31	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 =

16	20	22	31	33

$m = 22$

for processor P_2

18	19	25	26	34

$M_2 = 25$

for processor P_3

27	32	33	35	36

$M_3 = 33$

Step - VII

make a list of all medians

22	25	33
	↑ M	

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

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

$M = 25$

Step - VIII

~~Comparision~~

calculate $L =$ | 16 | 20 | 22 | 19 | 18 | $S_i < M$

5 element

$R =$ | 25 | $S_i = M$

one element $R = 1$ element

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

$U = 9$ element

Step - IX (for $K = 7$)

case-1 $|L| \geq |K| \Rightarrow 5 \geq 7$ - false

case-2 $|L| + |R| \geq K \Rightarrow 5 + 1 \geq 7$ - false

case-3 is executed.

$(U, K - (|L| + |R|))$ new $K = 7 - (|L| + |R|)$

process will continue
till the finding of the correct
case.