Q1. a. Models of Computation

Both sequential and parallel computers operate on a set (stream) 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 categories −

- Single Instruction stream, Single Data stream (SISD) computers
- Single Instruction stream, Multiple Data stream (SIMD) computers
- Multiple Instruction stream, Single Data stream (MISD) computers
- Multiple Instruction stream, Multiple Data stream (MIMD) computers

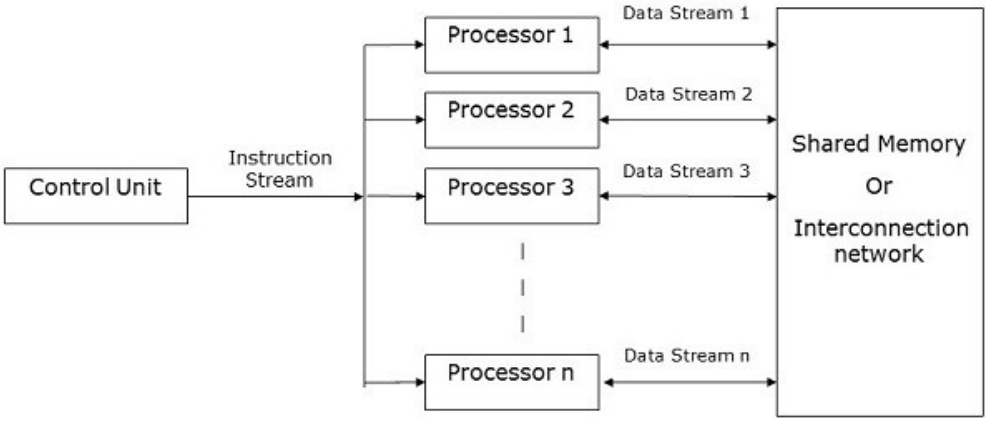
SISD Computers

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

In this type of computers, the processor receives a single stream of instructions from the control unit and operates on a 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 network**.



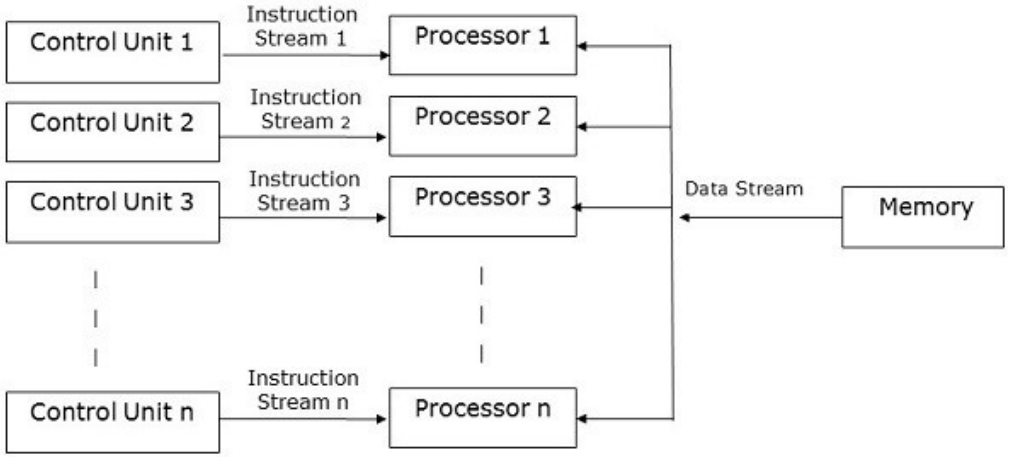
Here, one single control unit sends instructions to all processing units. During computation, at each step, all the processors receive a single set of instructions from the control unit and operate on different set of data from the memory unit.

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 network**.

While some of the processors execute a set of instructions, the remaining processors wait for their next set of instructions. Instructions from the control unit decides which processor will be **active** (execute instructions) or **inactive** (wait for next instruction).

MISD Computers

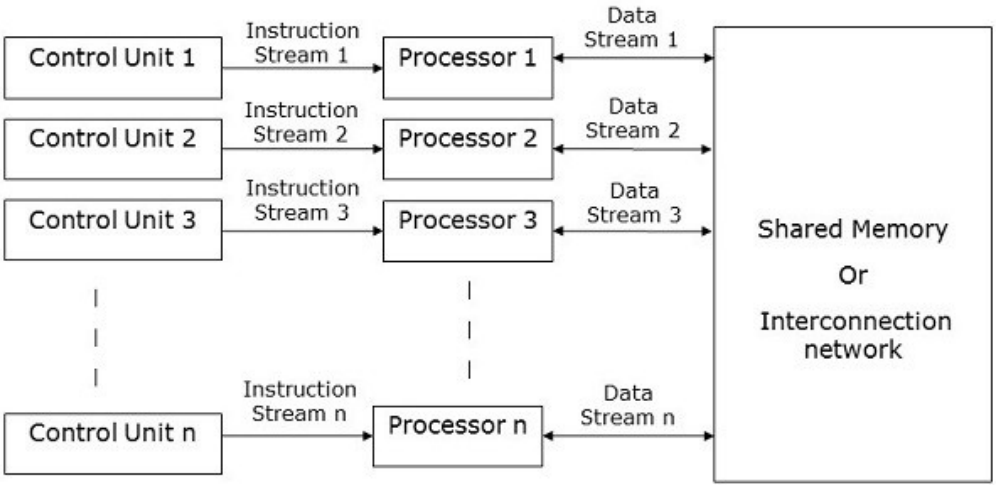
As the name suggests, MISD computers contain **multiple control units, multiple processing units,** and **one common memory unit**.



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 unit 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.

Q1. b. Scope of Parallel computing depends on 3 factors : Time complexity (Execution Time), Total number of processors used, and Total cost.

Time Complexity :

Worst-case complexity − When the amount of time required by an algorithm for a given input is maximum.

Average-case complexity − When the amount of time required by an algorithm for a given input is average.

Best-case complexity − When the amount of time required by an algorithm for a given input is minimum.

Number of Processors Used

The number of processors used is an important factor in analysing the efficiency of a parallel algorithm. The cost to buy, maintain, and run the computers are calculated. Larger the number of processors used by an algorithm to solve a problem, more costly becomes the obtained result.

Total Cost

Total cost of a parallel algorithm is the product of time complexity and the number of processors used in that particular algorithm.

Total Cost = Time complexity × Number of processors used

Therefore, the efficiency of a parallel algorithm is −

Worst case execution time of sequential algorithm

Efficiency =

Worst case execution time of the parallel algorithm

Q.1.c . The 15-puzzle problem is invented by sam loyd in 1878.

his problem there are 15 tiles, which are numbered from 0 – 15.

The objective of this problem is to transform the arrangement of tiles from initial arrangement to a goal

arrangement.

The initial and goal arrangement is shown by following figure.

There is always an empty slot in the initial arrangement.

The legal moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down.

Each move creates a new arrangement in a tile.

These arrangements are called as states of the puzzle.

The initial arrangement is called as initial state and goal arrangement is called as goal state.

The state space tree for 15-puzzle is very large because there can be 16! Different arrangements.

A partial state space tree can be shown in figure.

In state space tree, the nodes are numbered as per the level.

Each next move is generated based on empty slot positions.

Edges are labelled according to the direction in which the empty space moves.

The root node becomes the E – node.

The child node 2, 3, 4 and 5 of this E – node get generated.

Out of which node 4 becomes an E – node. For this node the live nodes 10, 11, 12 gets generated.

Then the node 10 becomes the E – node for which the child nodes 22 and 23 gets generated.

Finally, we get a goal state at node 23.

We can decide which node to become an E – node based on estimation formula.

Estimation formula:

C (x) = f (x) + G (x)

Where, f (x) is length of path from root to node x.

G (x) is number of non-blank tiles which are not in their goal position for node x.

C (x) denotes the lower bound cost of node x.

1. d . Algorithm for odd- even transposition sort :

Algorithm

```
procedure ODD-EVEN_PAR (n)

begin
   id := process's label

   for i := 1 to n do
   begin

      if i is odd and id is odd then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

      if i is even and id is even then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

   end for

end ODD-EVEN_PAR
```

1. E . Relevant differences with respect to the development of algorithms include the following.

(1) Reliability parameters. In wide-area networks the probability that something will go wrong during the transmission of a message can never be ignored; distributed algorithms for wide-area networks are usually designed to cope with this possibility. Local-area networks are much more reliable, and algorithms for them can be designed under the assumption that communication is completely reliable. In this case, however, the unlikely event in which something does go wrong may go undetected and cause the system to operate erroneously.

(2) Communication time. The message transmission times in wide-area networks are orders of magnitude larger than those in local-area networks. In wide-area networks, the time needed for processing a message can almost always be ignored when compared to the time of transmitting the message.

(3) Homogeneity. Even though in local-area networks not all nodes arenecessarily equal, it is usually possible to agree on common software and protocols to be used within a single organization. In wide-area networks a

variety of protocols is in use, which poses problems of conversion between different protocols and of designing software that is compatible with different standards.

(4) Mutual trust. Within a single organization all users may be trusted, but in a wide-area network this is certainly not the case. A wide-area network requires the development of secure algorithms, safe against offensive users at other nodes.

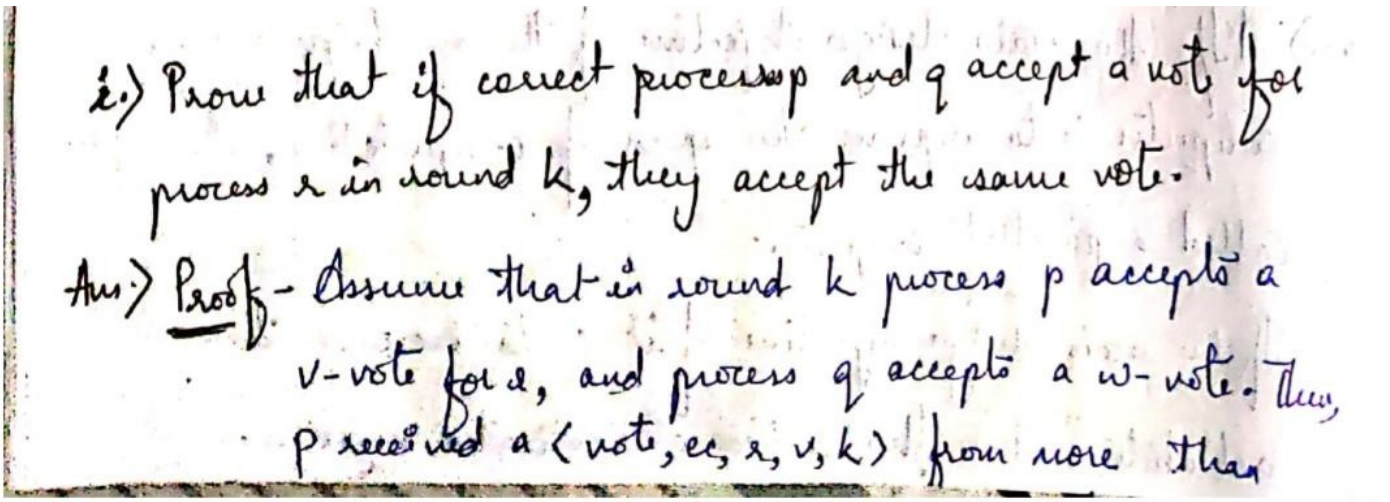
1. F . A multiprocessor computer is a computer installation that consists of several processors on a small scale, usually inside one large box. This type of computer system is distinguished from local-area networks by the following criteria. Its processors are homogeneous, i.e. , they are identical as hardware.The geographical scale of the machine is very small, typically of the order of one meter or less. The processors are intended to be used together in one computation (either to increase the speed or to increase the reliability) .If the main design objective of the multiprocessor computer is to improve the speed of computation, it is often called a parallel computer. If its main design objective is to increase the reliability, it is often called a replicated system.

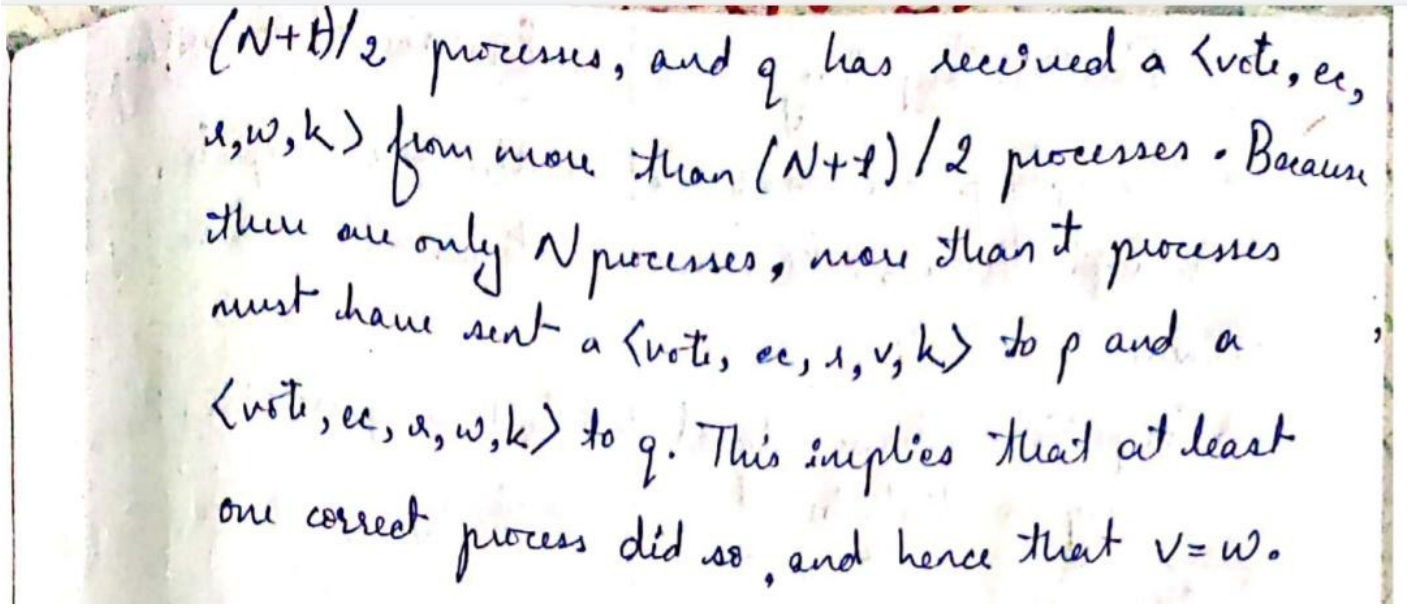
1. G . The goals of multiprocessor systems are: (i) to reduce the execution **time** of a single program (job) by decomposing it into processes, assigning processes to distinct processors and executing processes concurrently whenever possible; (ii) to increase the overall **throughput** of a system's work load by allowing several jobs to be processed simultaneously by the system;

1. H . Criteria for "good" routing methods include the following. (1) Correctness. The algorithm must deliver every packet offered to the network to its ultimate destination.

(2) Efficiency. The algorithm must send packets through "good" paths, e.g. , paths that suffer only a small delay and ensure high throughput of the entire network. An algorithm is called optimal if it uses the "best" paths. (3) Complexity. The algorithm for the computation of the tables must use as few messages, time, and storage as possible. Other aspects of complexity are how fast a routing decision can be made, how fast a packet can be made ready for transmission, etc., but these aspects will receive less attention in this chapter. (4) Robustness. In the case of a topological change (the addition or re-moval of a channel or node) the algorithm updates the routing tables in order to perform the routing function in the modified network. (5) Adaptiveness. The algorithm balances the load of channels and nodes by adapting the tables in order to avoid paths through channels or nodes that are very busy, preferring channels and nodes with a cur-rently light load. (6) Fairness. The algorithm must provide service to every user in the same degree.

1. I .





1. J . Robustness : In the case of a topological change (the addition or re- moval of a channel or node) the algorithm updates the routing tables in order to perform the routing function in the modified network.

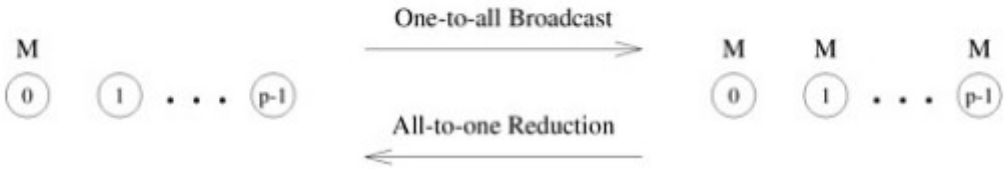
1. K .

PARALLEL COMPUTING VERSUS DISTRIBUTED COMPUTING

PARALLEL COMPUTING	DISTRIBUTED COMPUTING
Type of computation in which many calculations or the execution of processes are carried out simultaneously.	A system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.
Occurs in a single computer	Involves multiple computers
Multiple processors execute multiple tasks at the same time	Multiple computers perform tasks at the same time
Computer can have shared memory or distributed memory	Each computer has its own memory
Processors communicate with each other using a bus	Computers communicate with each other via the network
Increase the performance of the system	Allows scalability, sharing resources and helps to perform computation tasks efficiently

1. L . Parallel algorithms often require a single process to send identical data to all other processes or to a subset of them. This operation is known as one-to-all broadcast. Initially, only the source process has the data of size m that needs to be broadcast. At the termination of the procedure, there are p copies of the initial data one belonging to each process. The dual of one-to-all broadcast is all-to-one reduction. In an all-to-one reduction operation, each of the p participating processes starts with a buffer M containing m words. The data from all processes are combined through an associative operator and accumulated at a single destination process into one buffer of size m. Reduction can be used to find the sum, product, maximum, or minimum of sets of numbers the i th word of the accumulated M is the sum, product, maximum, or minimum of the i th words of each of the original buffers. Figure 4.1 shows one-to-all broadcast and all-to-one reduction among p processes.

Figure 4.1. One-to-all broadcast and all-to-one reduction.



1. M . Bitonic Sequence : A sequence is called Bitonic if it is first increasing, then decreasing. In other words, an array arr[0..n-i] is Bitonic if there exists an index i where 0<=i<=n-1 such that x0 <= x1 .....<= xi and xi >= xi+1..... >= xn-1

1. A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.

2. A rotation of Bitonic Sequence is also bitonic.

How to form a Bitonic Sequence from a random input?

We start by forming 4-element bitonic sequences from consecutive 2-element sequence. Consider 4-element in sequence x0, x1, x2, x3. We sort x0 and x1 in ascending order and x2 and x3 in descending order. We then concatenate the two pairs to form a 4 element bitonic sequence. Next, we take two 4 element bitonic sequences, sorting one in ascending order, the other in descending order (using the Bitonic Sort which we will discuss below), and so on, until we obtain the bitonic sequence.

Example: Convert the following sequence to bitonic sequence: 3, 7, 4, 8, 6, 2, 1, 5

Step 1: Consider each 2-consecutive elements as bitonic sequence and apply bitonic sort on each 2- pair elements.

In next step, take two 4 element bitonic sequences and so on.

Note: x0 and x1 are sorted in ascending order and x2 and x3 in descending order and so on

Step 2: Two 4 element bitonic sequences: A (3,7,8,4) and B (2,6,5,1) with comparator length as 2

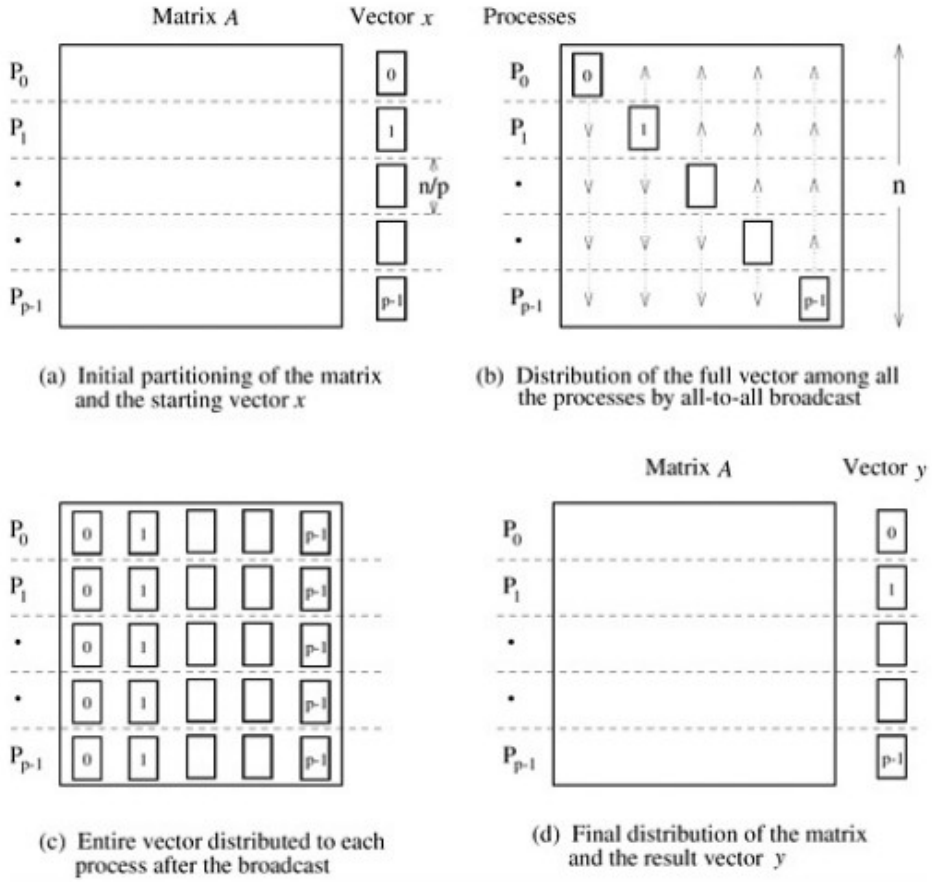
After this step, we'll get Bitonic sequence of length 8.

3, 4, 7, 8, 6, 5, 2, 1.

1. N . Same as 2.K.

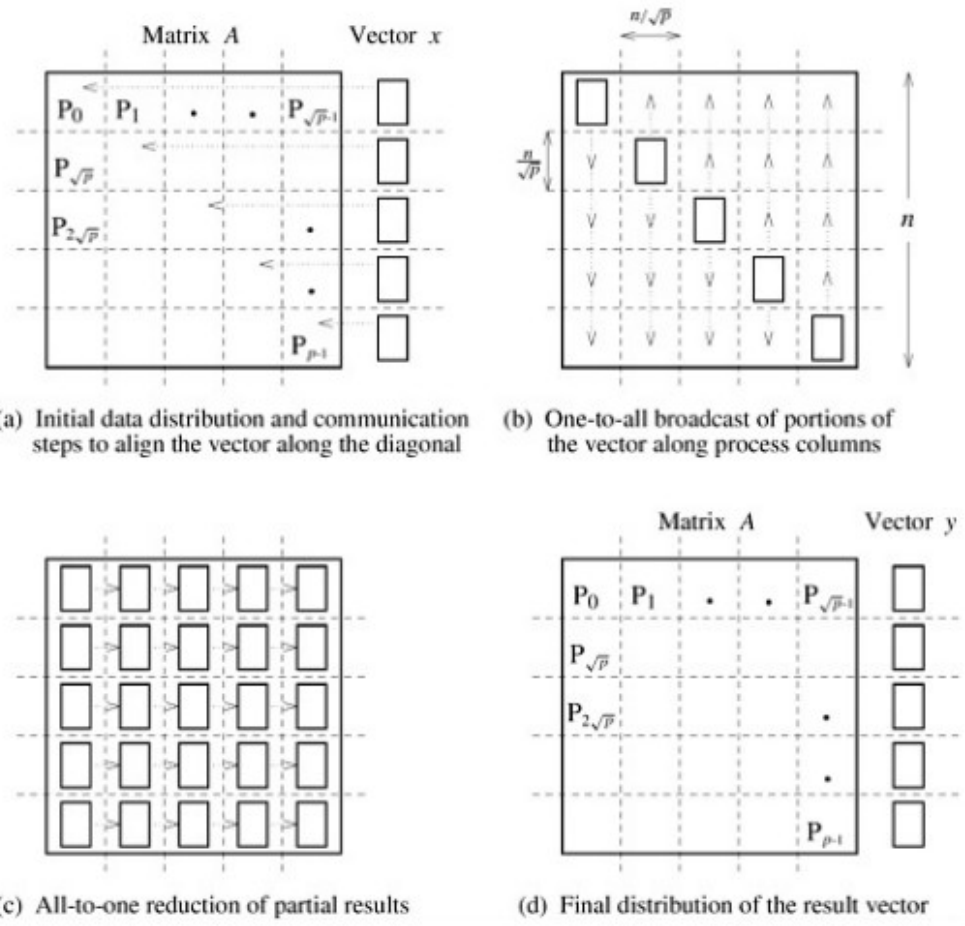
2. A. Matrix-vector multiplication using parallel processors :

1. USING Rowwise 1-D Partitioning : Multiplication of an n x n matrix with an n x 1 vector using rowwise block 1-D partitioning. For the one-row-per-process case, p = n.



One Row Per Process

2. USING 2-D Partitioning : Matrix-vector multiplication with block 2-D partitioning. For the one-element-per-process case, p = n2 if the matrix size is n x n.

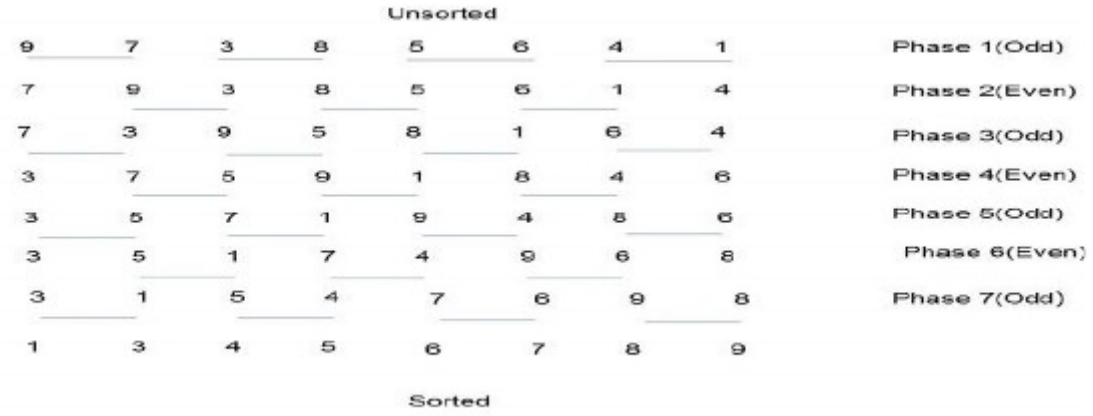


One Element Per Process

2. B. Same as 1a

2.c . same as 1b

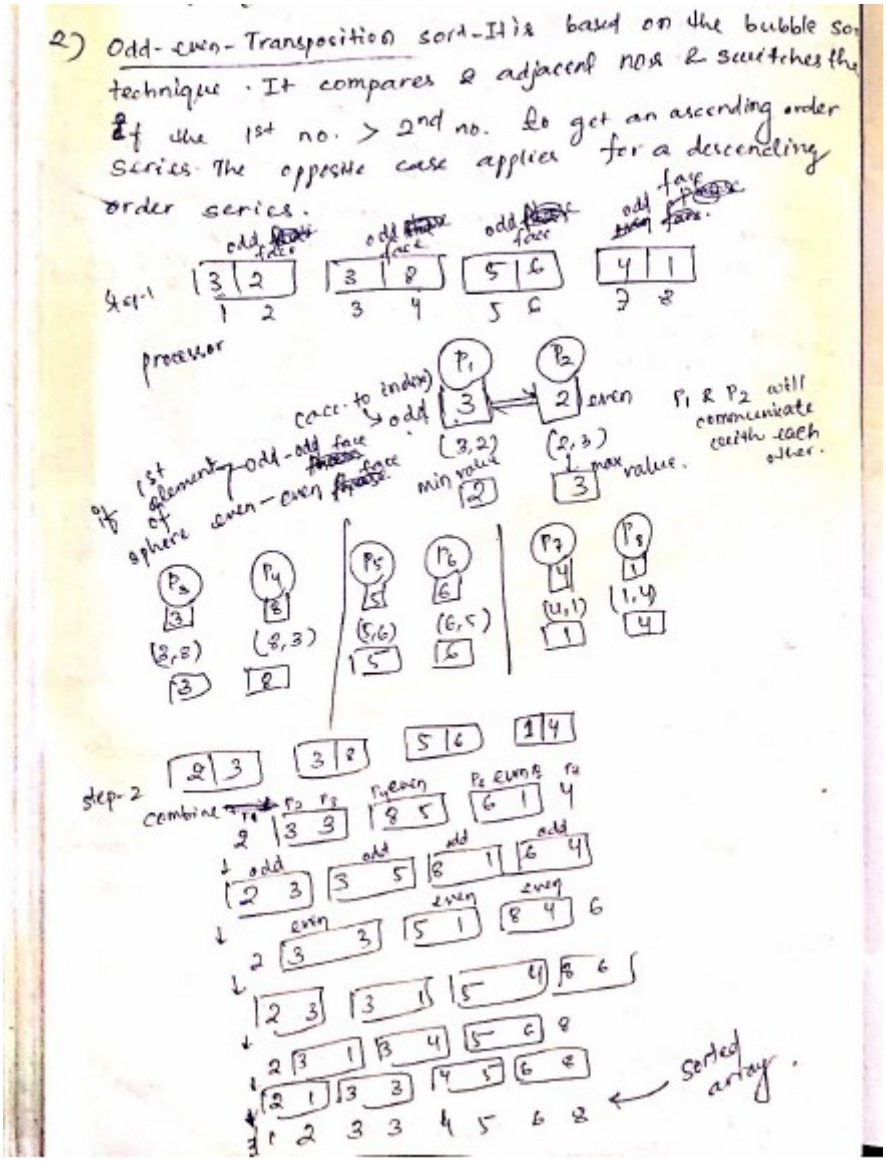
2.d. Odd-Even Transposition Sort : Odd-Even Transposition Sort is based on the Bubble Sort technique. It compares two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order list. The opposite case applies for a descending order series. Odd-Even transposition sort operates in two phases − odd phase and even phase. In both the phases, processes exchange numbers with their adjacent number in the right.



Algorithm

```
procedure ODD-EVEN_PAR (n)

begin
   id := process's label

   for i := 1 to n do
   begin

      if i is odd and id is odd then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

      if i is even and id is even then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

   end for

end ODD-EVEN_PAR
```

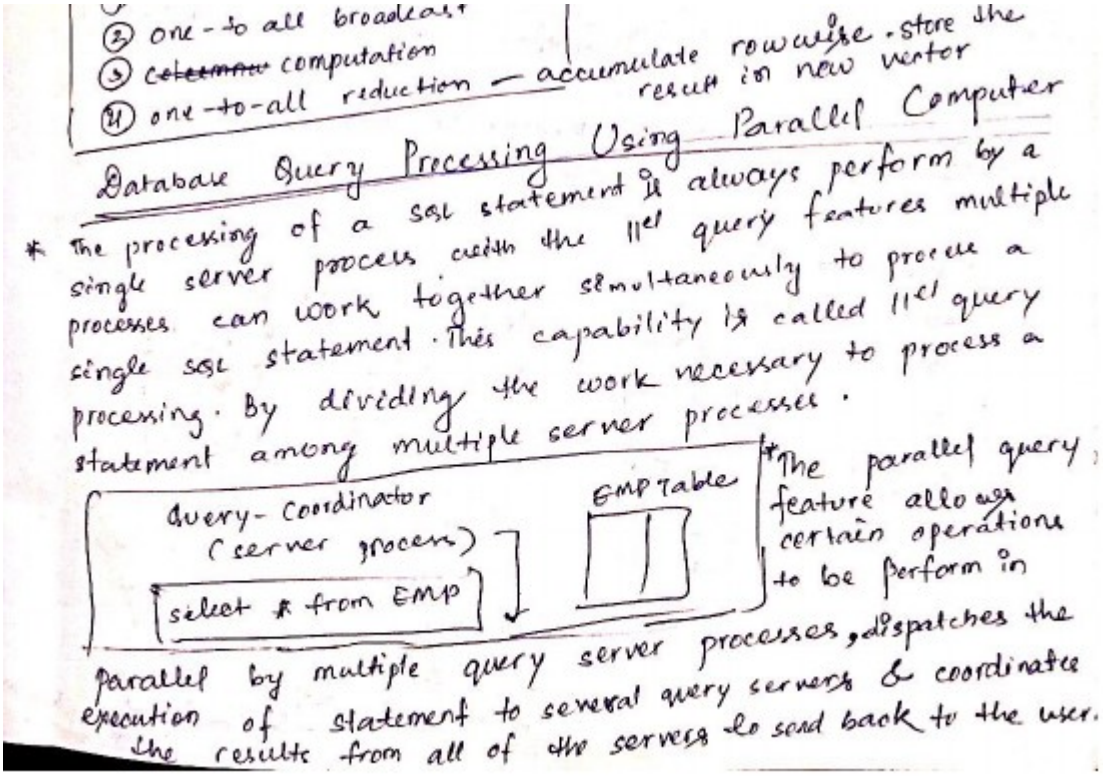
Example –

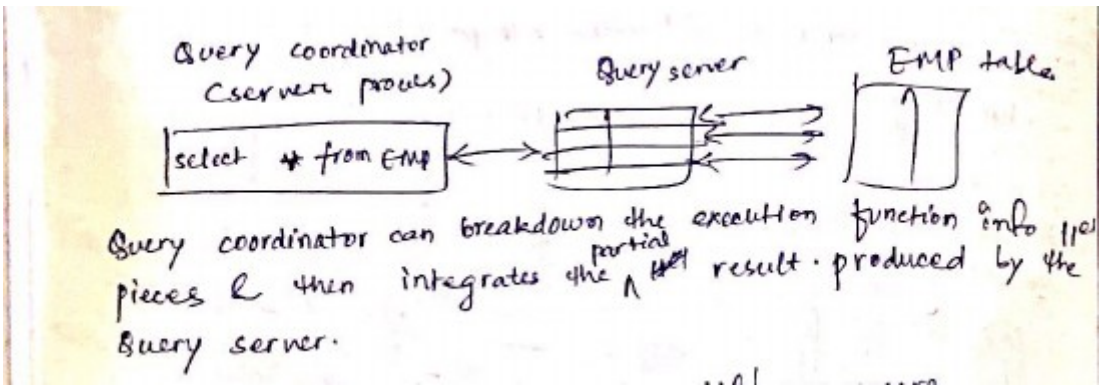


Time Complexity : To obtain a cost-optimal parallel formulation, we use fewer processes. Let p be the number of processes, where p < n. Initially, each process is assigned a block of n/p elements, which it sorts internally (using merge sort or quicksort) in Q((n/p) log(n/p)) time. After this, the processes execute p phases (p/2 odd and p/2 even), performing compare-split operations. At the end of these phases, the list is sorted (Problem 9.10). During each phase, Q(n/p) comparisons are performed to merge two blocks, and time Q(n/p) is spent communicating. Thus, the parallel run time of the formulation is :

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta(n)}^{\text{comparisons}} + \overbrace{\Theta(n).}^{\text{communication}}$$

2.E.





2.F. SORTING ON THE CREW MODEL :

Assume that a CREW SM SIMD computer with N processors PI, P2. . . , PN is to be used to sort the sequence S = {s1 s2 . . . , sn), where N < n.

Algorithm:-

procedure CREW SORT *(S)*

Step 1: for i = 1 to N do in parallel

Processor Pi

(1.1) reads a distinct subsequence Si of S of size n/N

(1.2) QUICKSORT (Si)

(1.3) S_i^1 <- Si

(1.4) P_i^1 <- Pi

end for.

O((n/N)log(n/N))

Cont...

Step 2 (2.1) u =1

(2.2) v = N

(2.3) while v > 1 do

(2.3.1) for m = 1 to |_v/2_| do in parallel

(i) P^{u+1}_m <- $P^u_{2m-1} \cup P^u_{2m}$

(ii) The processors in the set P^{u+1}_mperform

CREW MERGE (s^u_{2m-1}, s^u_{2m}, s^{u+1}_m)

end for

(2.3.2) if v is odd then

(1) $p^{U+1}_{v/2} = p^u_v$

(ii) $s^{U+1}_{v/2} = s^U_V$

end if

(2.3.3) u = u + 1

(2.3.4) V = v/2

end while.

O((n/N) + log n) time

Example

- Let S = (2, 8, 5, 10, 15, 1, 12, 6, 14, 3, 11, 7, 9, 4, 13, 16) and N = 4. Here N<n

Step1:- Subsequence S_i created : n/N=>16/4= 4

And Quick sort apply for sorting elements

S_1^1 ={2,5,8,10} S_2^1 = {1,6,12,15}

S_3^1= {3,7,11,14} S_4^1 = {9,13,14,16}

Step2:- u=1 & v=N=4

for (m=1 to v/2) 4/2=2

CREW MERGE ALGO USED

$P_1^2=p_1^1 \cup p_2^1$ =(p1,p2)=(1,2,5,6,8,10,12,15)

$P_2^2= p_3^1 \cup p_4^1$ =(p3,p4)=(3,4,7,9,11,13,14,16)

Cont....

The processors {P1, P2,P3, P4} cooperate to merge S_1^2 and s_2^2 into S_1^3 = (1, 2,. . . , 16) by using **CERW MERGE** .

Analysis:- the total running time of procedure CREW SOR'T is

t(n) = O((n/N)log(n/N)) + O((n/N)log N + log n log N)

= O((n/N)log n + log2n).

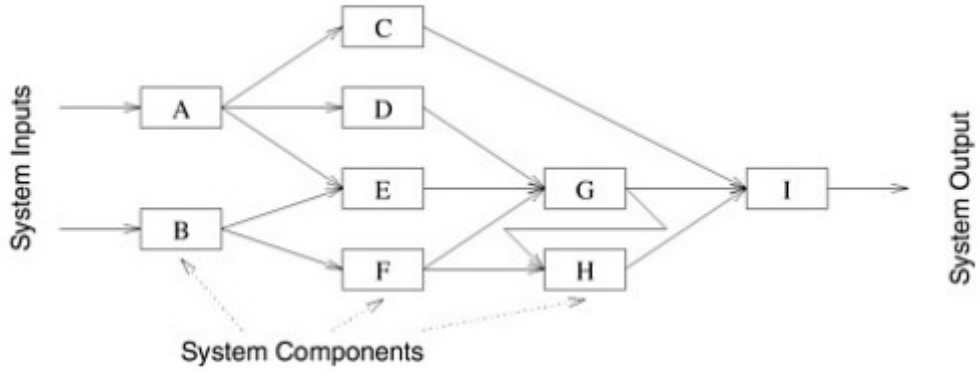
- Since p(n) = N, the cost is given by:-

c(n) = O(n log n + N log n^2).

2.G. Parallel discrete event simulation

Consider the simulation of a system that is represented as a network or a directed graph. The nodes of this network represent components. Each component has an input buffer of jobs. The initial state of each component or node is idle. An idle component picks up a job from its input queue, if there is one, processes that job in some finite amount of time, and puts it in the input buffer of the components which are connected to it by outgoing edges. A component has to wait if the input buffer of one of its outgoing neighbors if full, until that neighbor picks up a job to create space in the buffer. There is a finite number of input job types. The output of a component (and hence the input to the components connected to it) and the time it takes to process a job is a function of the input job. The problem is to simulate the functioning of the network for a given sequence or a set of sequences of input jobs and compute the total completion time and possibly other aspects of system behavior. Figure 3.20 shows a simple network for a discrete event solution problem.

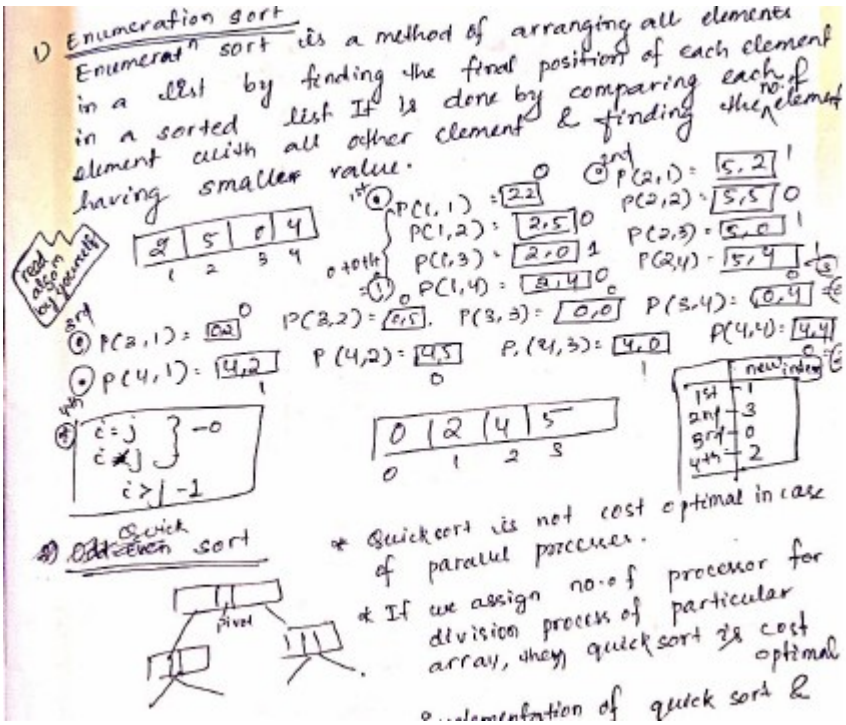
Figure 3.20. A simple network for discrete event simulation.



2.H. Enumeration sort is a method of arranging all the elements in a list by finding the final position of each element 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, for any two elements, ai and aj any one of the following cases must be true − **ai < aj , ai > aj , ai = aj**

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) then
         C[j] := 1;
      else
         C[j] := 0;

   for each process P1, j do
      A[C[j]] := A[j];

end ENUM_SORTING
```



Example :

2.I.

3 To prove (a) note that there is an element a_i $(0 < i < n/2)$ of sequence s such that for all $j < i$, $\min\{a_j, a_{n/2+j}\}$ belongs to $\{a_0, a_1, \ldots, a_{n/2-1}\}$ and for all $i < j < n/2$ $\min\{a_j, a_{n/2+j}\}$ belongs to $\{a_{n/2}, a_{n/2+1}, \ldots, a_{n-1}\}$. Similarly, for all $j < i$, $\max\{a_j, a_{n/2+j}\}$ belongs to $\{a_{n/2}, a_{n/2+1}, \ldots, a_{n-1}\}$ and for all $n/2 > j > i$, $\max\{a_j, a_{n/2+j}\}$ belongs to $\{a_0, a_1, \ldots, a_{n/2-1}\}$. Therefore,

$$s_1 = \{a_0, a_1, \ldots, a_i, a_{n/2+i+1}, \ldots, a_{n-1}\}$$

and

$$s_2 = \{a_{n/2}, a_{n/2+1}, \ldots, a_{n/2+i}, a_{i+1}, \ldots, a_{n/2-1}\}.$$

Note that both s_1 and s_2 are bitonic sequences.
Parts (b) and (c) can be proved in a similar way. For more detailed proof of the correctness of the bitonic split operation refer to Knuth [Knu73], Jaja [Jaj92], Quinn [Qui87], or Batcher [Bat68].

2.J. The implementation of a suitable communication system for these purposes requires the solution of the following algorithmical problems :

(1) The reliability of point-to-point data exchange. Two nodes connected by a line exchange data through this line, but they must somehow cope with the possibility that the line is unreliable. Due to atmospheric noise, power dips, and other physical circumstances, a message sent through a line may be received with some parts garbled, or even lost. These transmission failures must be recognized and corrected. This problem occurs not only for two nodes directly connected by a communication line, but also for nodes not directly connected, but communicating with the help of intermediate nodes. In this case the problem is even more complicated, because in addition messages may arrive in a different order from that in which they were sent, may arrive only after a very long period of time, or may be duplicated.

(2) Selection of communication paths . In a point-to-point network it is usually too expensive to provide a communication line between each pair of nodes. Consequently, some pairs of nodes must rely on other nodes in order to communicate. The problem of routing concerns the selection of a path (or paths) between nodes that want to communicate. The algorithm used to select the path is related to the scheme by which nodes are named, i.e. , the format of the address that one node must use to send a message to another node. Path selection in intermediate nodes is done using the address, and the selection can be done more efficiently if topological information is "coded" in the addresses.

(3) Congestion control. The throughput of a communication network may decrease dramatically if many messages are in transit simultaneously. Therefore the generation of messages by the various nodesmust be controlled and made dependent on the available free capacity of the network.

(4) Deadlock prevention . Point-to-point networks are sometimes called store-and-forward networks, because a message that is sent via several intermediate nodes must be stored in each of these nodes, then forwarded to the next node. Because the memory space available for this purpose in the intermediate nodes is finite, the memory must be managed carefully in order to prevent deadlock situations. In such situations, there exists a set of messages, none of which can be forwarded because the memory of the next node on its route is fully occupied by other messages.

(5) Security. Networks connect computers having different owners, some of whom may attempt to abuse or even disrupt the installations of others. Since it is possible to log on a computer installation from anywhere in the world, reliable techniques for user authentication, cryptography, and scanning incoming information (e.g. , for viruses) are required. Cryptographic methods can be used to encrypt data for security against unauthorized reading and to implement electronic signatures for security against unauthorized writing.

2.K. Quick Sort : Quicksort is one of the most common sorting algorithms for sequential computers because of its simplicity, low overhead, and optimal average complexity. Quicksort selects one of the entries in the sequence to be the pivot and divides the sequence into two - one with all elements less than the pivot and other greater. The process is recursively applied to each of the sublists.

COMPLEXITY OF QUICKSORT : O(n log n)

Quicksort algo….

procedure QUICKSORT (S)

if $|S| = 2$ and $s_2 < s_1$
then $s_1 \leftrightarrow s_2$
else if $|S| > 2$ **then**
(1) {Determine m, the median element of S}
SEQUENTIAL SELECT $(S, \lceil |S|/2 \rceil)$
(2) {Split S into two subsequences S_1 and S_2}
(2.1) $S_1 \leftarrow \{s_i : s_i \leqslant m\}$ and $|S_1| = \lceil |S|/2 \rceil$
(2.2) $S_2 \leftarrow \{s_i : s_i \geqslant m\}$ and $|S_2| = \lfloor |S|/2 \rfloor$
(3) QUICKSORT(S_1)
(4) QUICKSORT(S_2)
end if
end if. □

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 steps 3 and 4 in parallel until the dimension reaches 0.

Algorithm

```
procedure HYPERQUICKSORT (B, n)
begin

   id := process's 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 the process along the ith communication link;
         C := subsequence received along the ith communication link;
         B := B1 U C;
      endif

      else
         send B1 to the process along the ith communication link;
         C := subsequence received along the ith communication link;
         B := B2 U C;
         end else
      end for

   sort B using sequential quicksort;

end HYPERQUICKSORT
```

2.L. The physical layer (1) . The purpose of the physical layer in the IEEE standards is similar to that of the original ISO standard, namely to transmit sequences of bits. The actual standard definitions (the type of wiring, etc.) are, however, radically different, due t o the fact that all communication takes place via a commonly accessed medium rather than point-to-point connections. \

The medium access control sublayer (2a) . The purpose of this sublayer is to resolve conflicts that arise between nodes that want to use the shared communication medium. A statical approach would once and for all schedule the time intervals during which each node is allowed to use the medium. This method wastes a

lot of bandwidth, however, if only a few nodes have data to transmit, and all other nodes are silent; the medium remains idle during the times scheduled for the silent nodes. In token buses and token rings access to the medium is on a round-robin basis: the nodes circulate a privilege, called the token, among them and the node holding this token is allowed to use the medium. If the node holding the token has no data to transmit, it passes the token to the next node. In a token ring the cyclic order in which nodes get their turn is determined by the physical connection topology (which is, indeed, a ring) , while in a token bus the cyclic order is determined dynamically based on the order of the node addresses. In the carrier sense multiple access with collision detection (CSMA/CD) standard, nodes observe when the medium is idle, and if so they are allowed to send. If two or more nodes start to send (approximately) simultaneously, there is a collision, which is detected and causes each node to interrupt its transmission and try again at a later time.

The logical link control sublayer (2b) . The purpose of this layer is comparable to the purpose of the data-link layer in the OSI model, namely to control the exchange of data between nodes. The layer provides error control and flow control, using techniques similar to those used in the OSI protocols, namely sequence numbers and acknowledgements. Seen from the viewpoint of the higher layers, the logical link control sub layer appears like the network layer of the OSI model. Indeed, communi cation between any pair of nodes takes place without using intermediate nodes, and can be handled directly by the logical link control sublayer. A separate network layer is therefore not implemented in local-area networks; instead, the transport layer is built directly on top of the logical link control sublayer.

2.M. EREW Model : since N < n, N=n 1-x where 0<x<1. Now mi =[i(n/21/x)], for 1<=i<=21/x-1 . The mi can be used to divide S into 21/x subsequence of size n/21/x .These subsequences, denoted by S1,S2,..., Sj, Sj+1,.....S2j, where j =2(1/x)-1 . Every subdivision process can now be applied recursively to each of the subsequences Si until the entire sequence S is sorted in nondecreasing order. K= 2(1/x) .

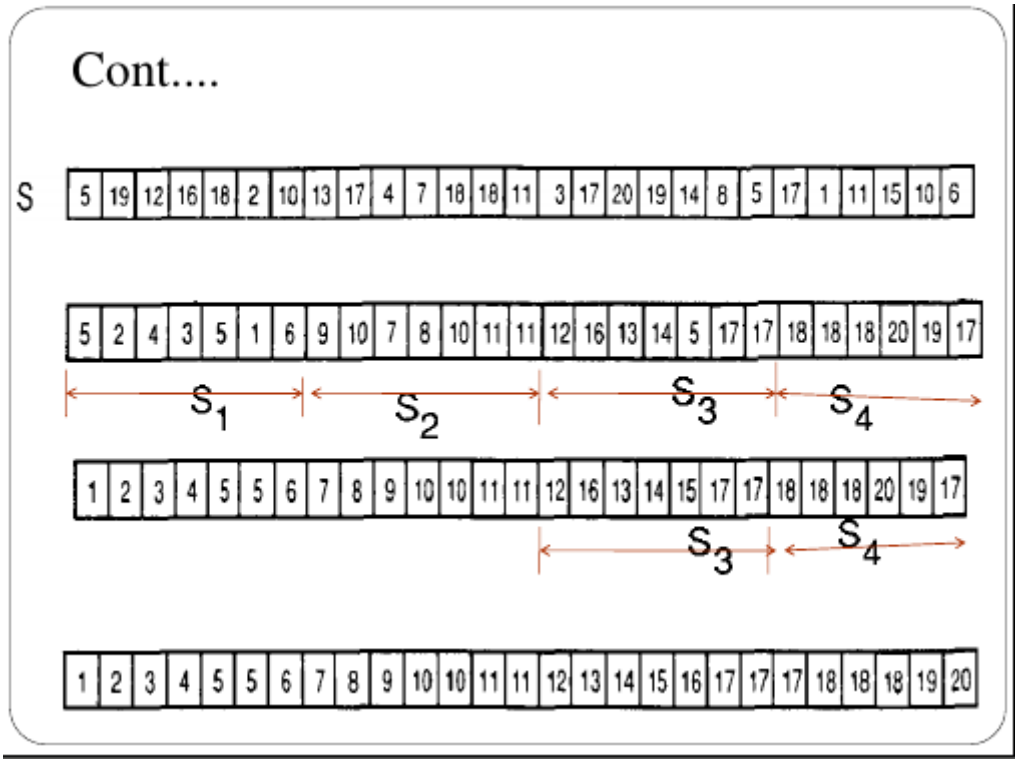
Example : Let S = {5,9, 12, 16, 18,2, 10, 13, 17,4,7, 18, 18, 11, 3, 17,20,19, 14, 8, 5, 17, 1, 11, 15, 10, 6) (i.e., n = 27) . Here N<n & N=n1-x => N=270.5 = 5 where 0<x<1 (x=0.5). K=21/x => k= 21/0.5 = 2 2 = 4 . During step 1 m1= 6 m2 = 11, and m3 = 17 are computed.The four sub sequences S1 ,S2, S3 and S4 are created. In step 5 the procedure is applied recursively and simultaneously to S1 and S2. Compute m1 = 2, m2= 4, and m3= 5, and the four subsequence {1,2}, {3,4}, {5,5), and (6) are created each of which is already in sorted order.

Algorithm:-

```
procedure EREW SORT (S)
Step1  if |S| < k
          then QUICKSORT (S)
       else (1) for i = 1 to k - 1 do
                   PARALLEL SELECT (S, |i |s|/k|)      [obtain mi]
                end for
            (2)  Si  = (s E S: s<=mi  )
            (3)  for i = 2 to k - 1 do
                      Si ={s E S : mi-1<=s <=mi }
                 end for
```

Cont..

```
            (4) Sk<= { s E S : s >=mk-1)
Step 2 for i = 1 to k/2 do in parallel
                EREW SORT (Si)
         end for
Step 3 for i = (k/2) + 1 to k do in parallel
                EREW SORT (Si)
          end for
          end if.
```



Running Time t(n) = cnx + 2t(n/k) = O(nx log n).

Since p(n) = n1-x, the procedure's cost is given by

c(n) = p(n) x t(n) = O(n log n), which is optimal.

2.N.

26 The CRCW PRAM algorithm can be easily adapted to other architectures by emulation.

CREW PRAM The main difference between a CREW and a CRCW PRAM architecture is the ability to perform a concurrent write. However, a concurrent write operation performed by n processors can be emulated in a CREW PRAM in $\Theta(\log n)$ time. Therefore, the parallel run time of the enumeration sort is $\Theta(\log n)$.

EREW PRAM In this model, in addition to requiring $\Theta(\log n)$ time to emulate a concurrent write operation, it needs a $\Theta(\log n)$ time to perform a concurrent read operation. Therefore, the parallel run time of the enumeration sort is $\Theta(\log n)$.

Hypercube The hypercube formulation is similar to the EREW PRAM formulation. Each read or write operation takes $\Theta(\log n)$ time. Also, the elements can be permuted to their final destination in $\Theta(\log n)$ time, since n^2 processors are used. Therefore, the parallel run time of the enumeration sort is $\Theta(\log n)$.

Mesh In a mesh-connected computer we can emulate a concurrent write or read operation in $\Theta(\sqrt{n})$ time. Therefore, the parallel run time of the enumeration sort is $\Theta(\sqrt{n})$.

When p processors are used, we assign n/p elements to each processor. Each processor now performs computations for each of its elements, as if a single element is assigned to it. Thus, each physical processor emulates n/p virtual processors. This results in a slowdown by a factor of n/p.

2.O. Distributed versus Centralized Algorithms : Distributed systems differ from centralized (uniprocessor) computer systems in three essential respects : (1) Lack of knowledge of global state. In a centralized algorithm control decisions can be made based upon an observation of the state of the system. Even though the entire state usually cannot be accessed in a single machine operation, a program may inspect the variables one by one, and make a decision after all relevant information has been considered. No data is modified between the inspection and the decision, and this guarantees the integrity of the decision. Nodes in a distributed system have access only to their own state and not to the global state of the entire system. Consequently, it is not possible to make a control decision based upon the global state. It is the case that a node may receive information about the state of other nodes and base its control decisions upon this information. In contrast with centralized systems, the fact that the received informa tion is old may render the information invalid, because the state of the other node may have changed between the sending of the state information and the decision based upon it. The state of the communication subsystem (Le . , what messages are in transit in it at a certain moment) is never directly observed by the nodes. This information can only be deduced indirectly by comparing information about messages sent and received by the nodes.

(2) Lack of a global time-frame. The events constituting the execution of a centralized algorithm are totally ordered in a natural way by their temporal occurrence; for each pair of events, one occurs earlier or later than the other. The temporal relation induced on the events constituting the execution of a distributed algorithm is not total; for some pairs of events there may be a reason for deciding that one occurs before the other, but for other pairs it is the case that neither of the events occurs before the other . Mutual exclusion can be achieved in a centralized system by requiring that if the access of process p to the resource starts later than the access of process q, then the access of process p must start after the access of process q has ended. Indeed, all such events (the starting and ending of the access of processes p and q) are totally ordered by the temporal relation; in a distributed system they are not , and the same strategy is not sufficient . Processes p and q may start accessing the resource, while the start of neither temporally precedes the start of the other.

(3) Non-determinism. A centralized program may describe the computation as it unrolls from a certain input unambiguously; given the program and the input, only a single computation is possible. In contrast, the execution of a distributed system is usually non deterministic, due to possible differences in execution speed of the system components.

2.P. The International Standards Organization (ISO) has fixed a standard for computer networking products such as those used (mainly) in wide-area networks. Their standard for network architectures is called the Open-Systems Interconnection (OSI) reference model, are as follows :

The OSI reference model consists of seven layers, namely the physical,data-link, network, transport, session, presentation, and application layers.The reference model specifies the interfaces between the layers and provides, for each layer, one or more standard protocols (distributed algorithms to implement the layer) .

The physical layer (1) . The purpose of the physical layer is to transmit sequences of bits over a communication channel. As the name of the layer suggests, this goal is achieved by means of a physical connection between two nodes, such as a telephone line, fiber optic connection, or satellite connection. The design of the layer itself is purely a matter for electrical engineers, while the 1/2 interface specifies the procedures by which the next layer calls the services of the physical layer. The service of the physical layer is not reliable; the bitstream may be scrambled during its transmission.

The data-link layer (2) . The purpose of the data-link layer is to mask the unreliability of the physical layer, that is, to provide a reliable link to the higher layers. The data-link layer only implements a reliable connection between nodes that are directly connected by a physical link, because it is built directly upon the physical layer. (Communication between non adjacent nodes is implemented in the network layer.) To achieve its goal, the layer divides the bitstream into pieces of fixed length, called frames. The receiver of a frame can check whether the frame was received correctly by verifying its checksum, which is some redundant information added to each frame. There is a feedback from the receiver to the sender to inform the sender about correctly or incorrectly received frames; this feedback takes place by means of acknowledgement messages. The sender will send a frame anew if it turns out that it is received incorrectly or completely lost. The general principles explained in the previous paragraph can be refined to a variety of different data-link protocols. For example, an acknowledgement message may be sent for frames that are received (positive acknowedgements) or for frames that are missing from the collection of receivedframes (negative acknowledgements) . The ultimate responsibility for thecorrect transmission of all frames may be at the sender's or the receiver'sside. Acknowledgements may be sent for single frames or blocks of frames,the frames may have sequence numbers or not , etc.

The network layer (3) . The purpose of the network layer is to provide a means of communication between all pairs of nodes, not just those connected by a physical channel. This layer must select the routes through the network used for communication between non-adjacent nodes and must control the traffic load of each node and channel. The selection of routes is usually based on information about the network topology contained in routing tables stored in each node. The network layer contains algorithms to update the routing tables if the topology of the network changes (owing to a node or channel failure or recovery) . Such a failure or recovery is detected by the data-link layer. Although the data-link layer provides reliable service to the network layer, the service offered by the network layer is not reliable. Messages (called packets in this layer) sent from one node to the other may follow different paths, causing one message to overtake the other. Owing to node failures messages may be lost (a node may go down while holding a message) , and owing to superfluous retransmissions messages may even be duplicated. The layer may guarantee a bounded packet

lifetime; i.e. , there exists a constant c such that each packet is either delivered at the destination node within c seconds, or lost .

The transport layer (4) . The purpose of the transport layer is to mask the unreliability introduced by the network layer, Le. , to provide reliable end-to-end communication between any two nodes. The problem would be similar to the one solved by the data-link layer, but it is complicated by the possibility of the duplication and reordering of messages. This makes it impossible to use cyclic sequence numbers, unless a bound on the packet lifetime is guaranteed by the network layer. The algorithms used for transmission control in the transport layer use similar techniques to the algorithms in the data-link layer: sequence numbers, feedback via acknowledgements, and retransmissions.

The session layer (5) . The purpose of the session layer is to provide facilities for maintaining connections between processes at different nodes. A connection can be opened and closed and between opening and closing the connection can be used for data exchange, using a session address rather than repeating the address of the remote process with each message. The session layer uses the reliable end-to-end communication offered by the transport layer, but structures the exchanged messages into sessions. A session can be used for file transfer or remote login. The session layer can provide the mechanisms for recovery if a node crashes during a session and for mutual exclusion if critical operations may not be performed at both

ends simultaneously.

The presentation layer (6) . The purpose of the presentation layer is to perform data conversion where the representation of information in one node differs from the representation in another node or is not suitable for transmission. Below this layer (Le. , at the 5/6 interface) data is in transmittable and standardized form, while above this layer (Le. , at the 6/7 interface) data is in user- or computer-specific form. The layer performs data compression and decompression to reduce the amount of data handed via the lower layers. The layer performs data encryption and decryption to ensure its confidentiality and integrity in the presence of malicious parties that aim to receive or corrupt the transmitted data.

The application layer (7) . The purpose of the application layer is to fulfill concrete user requirements such as file transmission, electronic mail, bulletin boards, or virtual terminals. The wide variety of possible applications makes it impossible to standardize the complete functionality of this layer, but for some of the applications listed here standards have been proposed.

2.Q.

Proof . Let V = {VI , . . . , VN } . We shall inductively construct a series of trees Ti = (Vi , Ei) (for i = 0, . . . , N) with the following properties.

(1) Each 1i is a subtree of G, i.e. , Vi $\subseteq$ V, Ei $\subseteq$ E, and 1i is a tree.

(2) Each Ti (for i < N) is a subtree of Ti+1 .

(3) For all i > 0, Vi E Vi and d E Vi .

(4) For all W E Vi , the simple path from w to d in Ti is an optimal path from w to d in G. These properties imply that TN satisfies the requirements for Td . To construct the sequence of trees, set Va = { d} and Eo = 0 . The tree 1i+1 is constructed as follows. Choose an optimal simple path P =(UO, . . . , Uk) from Vi+1 to d, and let 1 be the smallest index such that UI E Ti (such an 1 exists because Uk = d E Ti ; possibly 1 = 0) . Now set Vi+1 = Vi U { Uj : j < l} and Ei+1 = Ei U { (Uj , Uj +1) : j < l } . (The construction i s pictorially represented in Figure 4. 1 .) It i s easy to verify that 1i is a subtree of Ti+1 and that Vi+1 E Vi+1 . To see that Ti+1 is a tree, observe that by construction 1i+1 is connected, and the number of nodes exceeds the number of edges by one. (To has the latter property, and in each stage as many nodes as edges are added.) It remains to show that for all w E Vi+1 , the (unique) path from w to d in Ti+1 is an optimal path from w to d in G. For the nodes w E Vi C Vi+1 this follows because Ti is a subtree of 1i+1 ; the path from w to d in Ti+1 is the same as the path in 1i, which is optimal. Now let w = Uj , j < 1 be a node in Vi+1 \ Vi . Write Q for the path from UI to d in Ti , then in Ti+1 Uj is connected to d by the path (Uj , . . . , UI) concatenated with Q, and it remains. to show that this path is optimal in G. First, the suffix P' = (Ul ' . . . , Uk) of P is an optimal path from Ul to d, i.e. , C(P') = C(Q) : the optimality of Q implies C(P') 2: C(Q) , and C(Q) < C(P') implies (by the additivity of path costs) that the path (uo , . . . , Ul) concatenated with Q has lower cost than P, contradicting the optimality of P. Now assume that a path R from Uj to d has lower cost than the path (Uj , . . . , Ul) concatenated with Q. Then, by the previous observation, R has a lower cost than the suffix (Uj , . . . , Uk) of P, and this implies (again by the additivity of path costs) that the path (uo , . . . , Uj) concatenated with R has lower cost than P, contradicting the optimality of P.

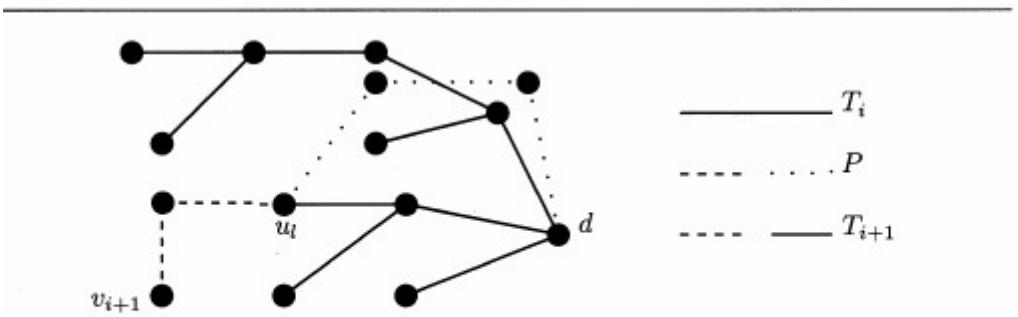


Figure 4.1 THE CONSTRUCTION OF T_{i+1}.

2.R. Definition : Let 8 be a subset of V. A path (uo , . . . , Uk) is an 8-path if for all i , 0 < i < k, Ui E 8. The 8-distance from u to v, denoted dB (u, v) , is the lowest weight of any 8-path from u to v (00 if no such path exists). The algorithm starts by considering aIl 0-paths, and incrementally computes 8-paths for larger subsets 8, until all V-paths have been considered. The following observations can be made.

Theorem 4.6 Algorithm 4.4 computes the distance between each pair of nodes in

8(N3) steps.

Proof. The algorithm starts with D [u, v] = 0 if u = v , D [u, v] = Wuv if uv E E and D[u, v] = 00 otherwise, and S = 0. Hence by Proposition 4.5, parts (1) and (2) , Vu, v : D [u, v] = dS (u, v) holds. In a pivot round

with pivot-node w the set S is expanded with w, and the assignment to D[u, v] ensures (by parts (3) and (4) of the proposition) that the assertion Vu, v : D [u, v] = dS (u, v) is preserved as a loop invariant . The program terminates when S = V, i.e. , (by parts (5) and (6) of the proposition and the loop invariant) the S-distances equal the distances. The main loop is executed N times, and contains N2 operations (which can be executed in parallel or serially) , which implies the time bound stated in the theorem.

2.S.

```
var Succ_p, Alive_p, Rcvd_p : sets of processes      init ∅ ;

begin shout ⟨ name, p ⟩ ;
      (* that is: forall q ∈ ℙ do send ⟨ name, p ⟩ to q *)
      while #Succ_p < L
            do begin receive ⟨ name, q ⟩ ; Succ_p := Succ_p ∪ {q} end ;
      shout ⟨ pre, p, Succ_p ⟩ ;
      Alive_p := Succ_p ;
      while Alive_p ⊈ Rcvd_p
            do begin receive ⟨ pre, q, Succ ⟩ ;
                     Alive_p := Alive_p ∪ Succ ∪ {q} ;
                     Rcvd_p := Rcvd_p ∪ {q}
               end ;
      Compute a knot in G
end
```

Algorithm 14.1 COMPUTATION OF A KNOT.

2.T.

```
var V_p : set of identities ;
    c_p : integer ;

begin V_p := {x_p} ; c_p := 0 ; shout ⟨ set, V_p ⟩ ;
      while true
      do begin receive ⟨ set, V ⟩ ;
               if V = V_p then
                  begin c_p := c_p + 1 ;
                        if c_p = N − t and y_p = b then
                           (* V_p is stable for the first time: decide *)
                           y_p := ( # V_p , rank(V_p, x_p) )
                  end
               else if V ⊆ V_p then
                  skip    (* Ignore "old" information *)
               else   (* new input; update V_p and restart counting *)
                  begin if V_p ⊂ V then c_p := 1 else c_p := 0 ;
                        V_p := V_p ∪ V ; shout ⟨ set, V_p ⟩
                  end
         end
end
```

Algorithm 14.2 A SIMPLE RENAMING ALGORITHM.

2.U. Assume, to the contrary, that such an algorithm, A, exists; a consensus algorithm A' may be derived from it, which constitutes a contradiction by Theorem 14.8. To simplify the argument we assume that CT contains two connected components, "0" and "1" . Algorithm A' first simulates A, but instead of deciding on value d, a process shouts (vote, d) and awaits the receipt of N - 1 vote messages. No deadlock arises, because all correct processes decide in A; hence at least N - 1 processes shout a vote message. After receiving the messages, process p holds N - 1 components of a vector in D N . This vector can be extended by a value for the process from which no vote was received, in such a way that the entire vector is in DT. (Indeed, a consistent decision was taken by this process, or is still possible.) Now observe that different processes may compute different extensions, but that these extensions belong to the same connected component of CT . Each process that has received N - 1 votes decides on the name of the connected component to which the extended vector belongs. It remains to show that A' is a consensus algorithm.

Termination. It has already been argued above that every correct process receives at least N - 1 votes.

Agreement. We first argue there exists a vector E DT such that each correct process obtains N - 1 components of d.

Case 1 : All processes found a decision in A. Let d_0 be the vector of decisions reached; each process obtains N - 1 components of d d_0 , though the "missing" component may be different for each process.

Case 2: All processes except one, say r, found a decision in A. All correct processes receive the same N - 1 decisions, namely those of all processes except r . It is possible that r crashed, but because it is also possible that r is only slow, it must still be possible for r to reach a decision, i.e. , there exists a vector d_0 E DT which extends the decisions taken so far. From the existence of d_0 it follows that each process decides on the connected component of this vector d_0.

Non-triviality. By the non-triviality of A, decision vectors in both component 0 and component 1 can be reached; by the construction of A', both decisions are possible.

Thus, A' is an asynchronous, deterministic, 1-crash-robust consensus algorithm. With Theorem 14.8 the non-existence of algorithm A follows.

2.V. two sets of processes S and T can be formed such that the processes in each group decide independently of the other group. If one group can reach a decision in which there is a leader, an execution can be constructed in which two leaders are elected. If one group can reach a decision in which no leader is elected, an execution can be constructed in which all processes are correct, but no leader is elected.