Objective

Q1: Fill in the blanks

1. Cluster
2. Size
3. MPI_Ssend, recv
4. Simple instruction, multiple data
5. Local Area Multicomputer
6. _np
7. Graphics Processing Unit
8. MPI_Barrier
9. Parallel Random Access Machine
10. Don't know (lg n)
11. EREW exclusive read exclusive write
12. P(n) x T(n)
13. MIMD multiple instruction multiple data
14. Cuda, Libra
15. Don't know
16. MISD multiple instruction single data

Q2: True/False

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

4 11 12 are not confirmed

Q3: choose the best answer

1. MPI_Ssend, MPI_SRecv
2. MIMD
3. SIMD
4. Gather
5. Scatter
6. MIMD
7. SIMD
8. SISD
9. Don't know
10. Don't know

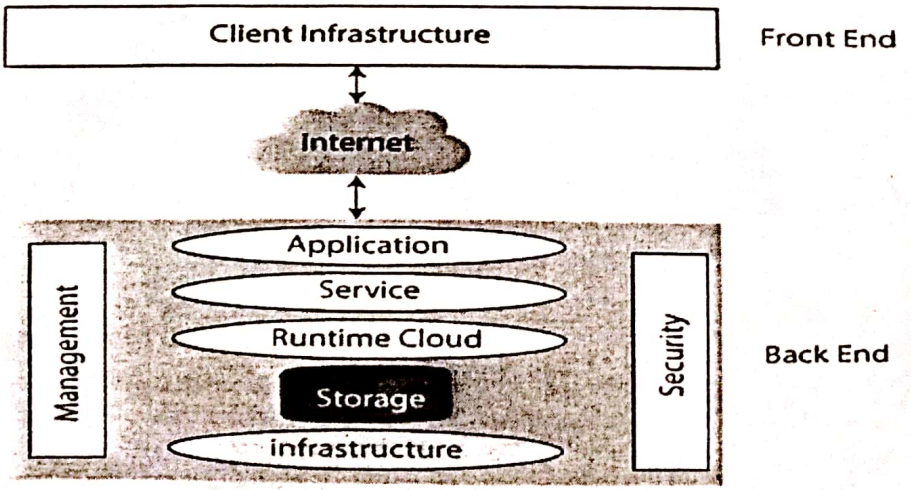
Subjective

Q1: what is parallel computing and why we need it ?

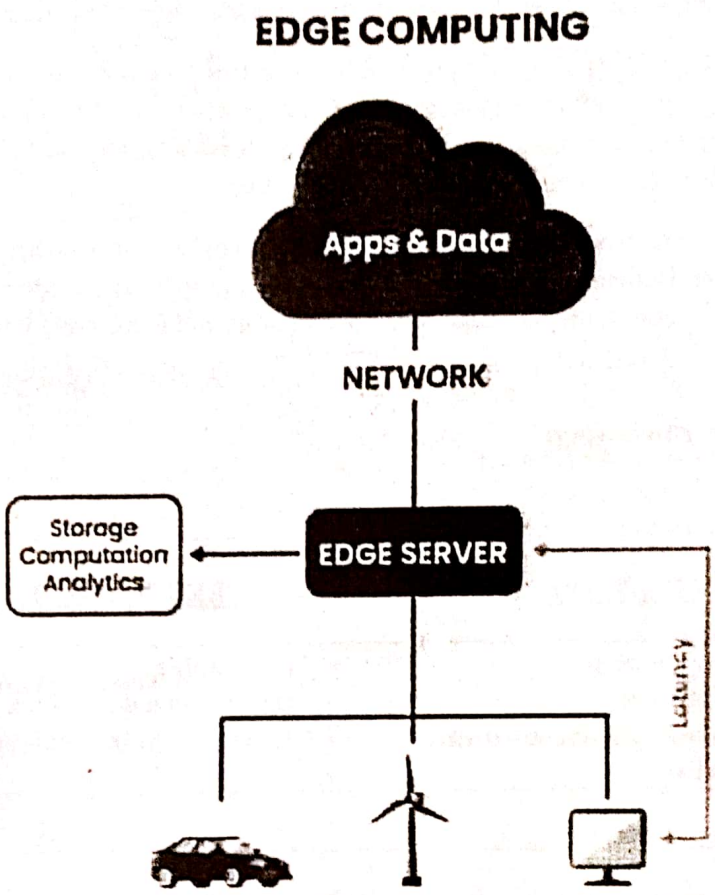
Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). We need parallel computing because we can use multiple processors in parallel to solve problems more quickly than with a single processor.

Q2: Briefly describe the working of the following with the help of diagram.

CLOUD COMPUTING(2019) Cloud Computing is the delivery of Different Services Through Internet. Include tools and application data Storage, Server, Database Networking & software .

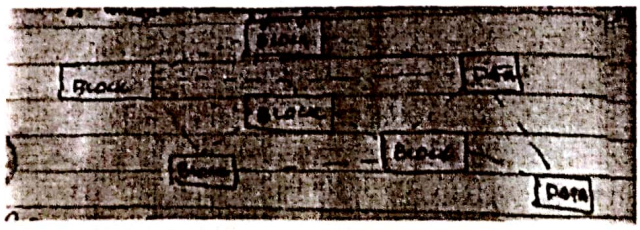


EDGE COMPUTING(2019): Edge computing is a type of computing that takes place at or near the edge of a network. The processing occurs either within or close to the device, so less data travels to the central server



BLOCK CHAIN(2019): It Consist of Multiple block of data stuck together when a new block stores a new data it is added to the blockChain

- Transaction must require
- Transaction must be verified with the public record of information
- That transcation must be stored in that block
- Block must given a hash which is unique identifying code called hash



ARRAY PROCESSOR(2014): Array processor is a synchronous parallel computer with multiple ALU called processing elements (PE) that can operate in parallel in lock step fashion. It is composed of N identical PE under the control of a single control unit and many memory modules. Array processor also frequently use a form of parallel computation called pipelining where an operation is divided into smaller steps and the steps are performed simultaneously.

VECTOR PROCESSOR(2014): A vector processor is a central processing unit that can work on an entire vector in one instruction. The instruction to processor is in the form of one computer vector instead of its elements. It is also known as an array processor. It exhibits SIMD behavior by having operations that are applied to all elements on vector.

GPU(2014): A graphics processing unit (GPU) is a single drip processor primarily used to manage and boost the performance of video and graphics. It is in PCs on a video card or mother board as well as mobile phones, display adapters, work stations and game consoles.

Q3(a): Differentiate the following:

CIIENT/SERVER APP	PEER TO PEER APP
• More stable and Secure • Centralized Network • Doesnot require expensive hardware to setup the network	• Less Stable • Decentralized Network • Expensive to implement

Mutual Exclusion(Mutax)	Synchorization(using Condition Variable)
• Mutual exclusion is a program object that prevent simethanously acess to shared resource. This concept use in concurrent programming with a critical section, a piece of code in which processor or thread access a shared	• Condition variables are synchronization objects that allow threads to wait for certain events (conditions) to occur. Condition variables are slightly more complex than mutexes, and the correct use of condition variables requires the thread to co-operatively use a specific protocol in order to ensure safe and consistent serialization

CLUSTER COMPUTING	GRID COMPUTING
• It is homogenous network. Similar hardware component running a similar operating system are connected together in a cluster. • Services Oriented • They are within the same location or complex.	• It is a heterogenous network. Different computer hardware running various kinds of operating systems are connected together in a grid. • Application Oriented • They are distributed over a LAN, MAN or WAN. They can be geographically separated.

MULTIPROCESSOR	MULTICOMPUTER
• Support Parallel Computing • More Diffcuilt and costly to build • Easier to Process	• Support Distributed Computing • Easier and Cost effective to build • Less easy to Program

Q3(b): Mark where applicable.

	TCP	UDP	MPI
UNICASTING	yes	yes	yes
MULTICASTING	no	yes	yes
BROADCASTING	no	yes	yes
MANY TO ONE	no	no	yes
MANY TO MANY	no	no	yes

Q4(2014): Write an MPI program to generate following cartesian topology.

ODD EVEN TOPOLOGY

```
#include "mpi.h"
void main(int argc, char *argv[])
{
    int nrow, mcol, root, Iam, ndim, p, rank;
    int dims[2], coords[2], cyclic, reorder;
    MPI_Comm comm, comm1, ceven, codd;
    MPI_Group e_group, o_group;

    MPI_Init(&argc, &argv);            /* starts MPI */
    MPI_Comm_rank(MPI_COMM_WORLD, &Iam);  /* get current process id */
    MPI_Comm_size(MPI_COMM_WORLD, &p);    /* get number of processes */

    nrow = 4; mcol = 2; ndim = 2;
    root = 0; cyclic = 1; reorder = 1;

    dims[0] = nrow;     /* rows */
    dims[1] = mcol;     /* columns */

    for(int i=0; i<p/2; i++)

    {
       ranks[i]= i*2;
     }

    MPI_Group world_group;
    MPI_Comm_group(MPI_COMM_WORLD, &world_group);

    MPI_Group_incl(world_group, n/2, ranks, &e_group);
    MPI_Group_excl(world_group, n/2, ranks, &o_group);

    MPI_Comm_create_group(MPI_COMM_WORLD, o_group, 0, &comm);
    MPI_Comm_create_group(MPI_COMM_WORLD, e_group, 0, &comm1);

    MPI_Cart_create(comm, ndim, dims, cyclic, reorder, &codd);
    MPI_Cart_create(comm1, ndim, dims, cyclic, reorder, &ceven);

    MPI_Finalize();
}
```

Q5(2014) Q6(2019):

Matrix addition

```
#include "mpi.h"
#define row_size = 5
#define col_size = 3

void main(int argc, char *argv[])
{
      int matrix1[row_size * col_size];
      int matrix2[row_size * col_size];
      int matrix3[row_size * col_size];

      MPI_Init(&argc, &argv);            /* starts MPI */
      MPI_Comm_rank(MPI_COMM_WORLD, &Iam);  /* get current process id */
      MPI_Comm_size(MPI_COMM_WORLD, &p);    /* get number of processes */

      int row1[col_size];
      int row2[col_size];
      int row3[col_size];

     //matrix1 , col_size, MPI_INT -- for master
     //row1, col_size, MPI_INT -- for other

     //for first matrix
       MPI_Scatter(matrix1 , col_size, MPI_INT, row1, col_size, MPI_INT, 0,
MPI_COMM_WORLD);
     //for second matrix
       MPI_Scatter(matrix2 , col_size, MPI_INT, row2, col_size, MPI_INT, 0,
MPI_COMM_WORLD);

      sum(row1, row2, &row3);

      MPI_Gather(row3, col_size, MPI_INT, matrix3, col_size*row_size, MPI_INT, 0,
MPI_COMM_WORLD);
      if( Iam== 0){
         for(int i=0; i< row_size*col_size; i++){
            printf("%d", matrix3[i]);
         }
      }

      MPI_Finalize();
}
```

Q6(2014) Q5(2019):

SUM OF NUMBERS

```
#include "mpi.h"
#define SIZE = 1000

void main(int argc, char *arg v[])
{
int data[SIZE];
MPI_Init(&argc, &argv);              /* starts MPI */
MPI_Comm_rank(MPI_COMM_WORLD, &Iam);  /* get current process id */
MPI_Comm_size(MPI_COMM_WORLD, &p);    /* get number of processes */

MPI_Bcast(data, SIZE, MPI_INT,0, MPI_COMM_WORLD);

int s=sum(data, Iam*SIZE/p, SIZE/p);
int asum=0;

MPI_Reduce(&s, &asum, 1, MPI_INT, MPI_SUM, 0,MPI_COMM_WORLD);

if(Iam == 0){
      printf("%d", asum);
     }
MPI_Finalize();

}

void sum(int[] data, int start, int range)
{
 int ans=0;
 for(int i=start; i<start + range; ++i)
  {
     ans += data[i];
  }
return ans;
}
```

Q7 in 2014 and Q4 2019:

Algorithm Broadcast_EREW

Processor P_1

y (in P_1's private memory) ← x

L[1] ← y

For i=0 to log p - 1 **do**

Forall P_j, where $2^i + 1 \leq j \leq 2^{i+1}$ do in parallel

y (in P_j's private memory) ← L[$j - 2^i$]

L [j] ← y

endfor

endfor

Complexity Analysis

Run Time → T(n)
Number of Processors → P(n)
Cost → C(n) = T(n) * P(n)

Q8:

Algorithm Broadcast_EREW

For i=0 to log n **do**

Forall P_j, where $2^{i-1} + 1 \leq j \leq n$ do in parallel

A [j] ← A [j] + A [$j - 2^{i-1}$]

endfor

endfor

Complexity Analysis

Run Time → T(n)
Number of Processors → P(n)
Cost → C(n) = T(n) * P(n)