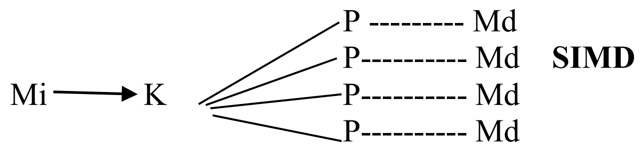
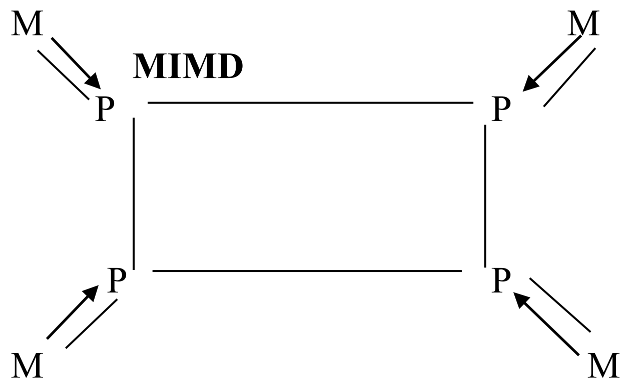
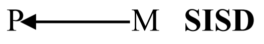
Q1. (i) What are the two fundamental types or parallelism ?
(ii) What are the categories into which parallel computers are divided according to Flynn's taxonomy ?

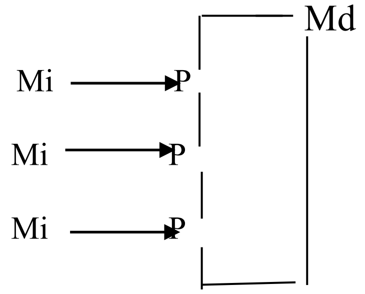
Ans (i) Data and task parallelism
Ans. (ii) Single Instruction Single data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), Multiple Instruction Multiple Data (MIMD)

(a) 

(b) 

(c) 

MISD

(d) 

Q2. Explain why or why not the loop below can be executed safely in parallel using loop parallelism?

C pseudo code :
```
for ( I = 1; I < 1000000 +1 ; I++ ) {
        Y[I] = X[I-1] + Y[I] + Z[I+1] ;
}
```

Ans 2. No loop dependencies (i.e. loop iteration is not recursively dependent on previous iteration), hence loop can be executed in parallel.

Q3.(i) If F_s is the serial fraction of a code prove that the upper bound on speedup is $1/_{Fs}$ no matter how many processors you use.
(ii) If the operations in a code are 90% parallel, what is the maximum speedup on a 9 processor parallel machine? (Hint Amdahl's Law)

Ans 3 (i)

$$\text{Speedup} = \frac{1}{F_s + F_p/N}$$

Here Fp is the parallel fraction of the code and N is the # of processors. Let N go to infinity and we have speedup = $1/F_s$

3(ii) speed up = (1. / (0.1 + .9/9)) = 1. / (.2) = 5

Q4. What is wrong in the pseudo code below assuming that it wants to call work with variable I where I goes from 0 to np-1? Explain in words how you would correct that?

C version:

```
np = omp_get_num_threads() ;
#pragma omp parallel for schedule(static)
  for (I=0; I<np; I++)
    work(I);
```

Ans 4. omp_get_num_threads() is only valid within a parallel construct. So it needs to be moved inside the parallel loop.

Q5. Below is a correct serial pseudo code (choose either Fortran or C code)

```
void ccode( float a[], float b[],
float c[], int n)
{    float x, y ;
x, y
   int i;
   for (i = 0 ; i < n ; i++ ) {
      x = a[i] - b[i] ;
      y = b[i] + a[i] ;
      c[i] = x * y ;
   }
 }
```

```
subroutine fcode
integer n
real a(n), b(n), c(n),

do i = 1,n
     x=a(i) - b(i)
y=b(i) + a(i)
c(i) = x * y
end do
end
```

The codes are parallelized using OpenMP as follows :

```
void ccode( float a[], float b[],
float c[], int n)
{   float x, y ; int i;
#pragma omp parallel for \
shared(a,b,c,n,x,y) private(i)


    for (i = 0 ; i < n ; i++ ) {
        x = a[i] - b[i] ;
        y = b[i] + a[i] ;
        c[i] = x * y ;
    }
}
```

```
subroutine fcode
integer n
real a(n), b(n), c(n), x, y
!$omp parallel do shared(a,b,c,n,x,y)
!$omp private(i)

do i = 1,n
        x=a(i) - b(i)
        y=b(i) + a(i)
        c(i) = x * y
end do
!$omp end parallel do
```

What is wrong in the parallel OpenMP code that will prevent it from producing correct result like the serial code and how will you correct that?

Ans 5. We need to declare x and y to be private.

Q6. If the following code fragment is run on 4 processors, how many times will each print statement be executed:

C :

```
printf("Print # 1\n");
#pragma omp parallel
{
printf("Print #2\n");
#pragma omp for
for (i=0;i<40;i++) {
   printf("Print #3\n);
   }
printf("Print #4\n");
}
```

Ans 6. 'Print #1' will be printed once. 'Print #2' will be printer 4 times. 'Print #3' will be prited 40 times.'Print #4' will be printed 4 times.

Q7. What is the main difference between point to point communications and collective communications in MPI?

Ans 7. In point to point communication only two processors take part. Collective communication can be from one to many, many to one, or many to many processors.

Q8.i. Give a pseudo code for unidirectional communication.

Ans 8.
```
If (myrank == 0) then
               call MPI_send( .,., …)
   elseif (myrank == 1) then
               call MPI_Recv(.,.,…)
    endif
```

Q8.ii. Name a collective communication call in MPI where only data is transferred.

Ans 8. BCast

Q8.iii. Name a collective call in MPI where some mathematical operation is done.

Ans 8. Reduce

Q9. Three processors have the following data in an array in each processor:

Proc0	Proc1	Proc2
a(1) = 1	a(1) = 4	a(1) = 7
a(2) = 2	a(2) = 5	a(2) = 8
a(3) = 3	a(3) = 6	a(3) = 9

After a specific MPI routine call the data gets distributed as follows:

Proc0	Proc1	Proc2
b(1) = 1	b(1) = 2	b(1) = 3
b(2) = 4	b(2) = 5	b(2) = 6
b(3) = 7	b(3) = 8	b(3) = 9

What is this MPI routine called?

Ans 9. MPI_AlltoAll ( )