1.Given a nxn matrix A and a vector x of length n, their product y=A·x, write a program to implement the multiplication using OpenMP PARALLEL directive.

```
int main() {
   int i, j, n;
   printf("Enter the size of the matrix and vector (n): ");
   scanf("%d", &n);
   double A[n][n], x[n], y[n];
   printf("Enter the elements of the %dx%d matrix A:\n", n, n);
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%lf", &A[i][j]);
      }
   }
   printf("Enter the elements of the vector x of size %d:\n", n);
   for (i = 0; i < n; i++) {
      scanf("%lf", &x[i]);
   }
   for (i = 0; i < n; i++) {
      y[i] = 0;
   }
   #pragma omp parallel for private(j)
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         y[i] += A[i][j] * x[j];
      }
   }
   printf("Resultant vector y:\n");
   for (i = 0; i < n; i++) {
        printf("%.2f\n", y[i]);
    }

    return 0;
}
```

2. Consider a Scenario where a person visits a supermarket for shopping. He purchases various items in different sections such as clothing, gaming, grocery, stationary. Write anopen MP program to process his bill parallelly in each section and display the final amount to be paid. (sum of elements parallelly).

```
#include <stdio.h>
#include <omp.h>
int main() {
    int i, sections;
    printf("Enter the number of sections in the supermarket: ");
    scanf("%d", &sections);

    double bills[sections], total = 0.0;
    printf("Enter the amounts spent in each section:\n");
    for (i = 0; i < sections; i++) {
        printf("Section %d: ", i + 1);
        scanf("%lf", &bills[i]);
    }
    #pragma omp parallel for reduction(+:total)
    for (i = 0; i < sections; i++) {
        total += bills[i];
    }
    printf("Final amount to be paid: %.2f\n", total);
    return 0;
}
```

3. X on the earth, to find his accurate position on the globe werequire the value of Pi. Writea program to compute the value of pi function by NumericalIntegration using OpenMP PARALLEL section.

```
#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

static long num_steps;

double step;

int main() {

   int i;

   double x, pi, sum = 0.0;

   printf("Enter number of steps: ");

   scanf("%ld", &num_steps);

   if (num_steps <= 0) {

      printf("Number of intervals should be a positive integer.\n");

      exit(1);

   }

   step = 1.0 / (double)num_steps;

   #pragma omp parallel for private(x) reduction(+:sum)

   for (i = 1; i <= num_steps; i++) {

      x = (i - 0.5) * step;

      sum = sum + 4.0 / (1.0 + x * x);

   }

   pi = sum * step;

   printf("Estimated value of Pi = %.8f\n", pi);

   return 0;

}
```

4. Using OpenMP, Design and develop a multi-threaded program to generate and printFibonacci Series. One thread must generate the numbers up to the specified limit and another thread must print them. Ensure proper synchronization.

```
#include <stdio.h>

#include <omp.h>

#define MAX_SIZE 100

int fibonacci[MAX_SIZE];

void generate_fibonacci(int n) {

   fibonacci[0] = 0;

   fibonacci[1] = 1;


   #pragma omp parallel for

   for (int i = 2; i < n; i++) {

      fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];

   }

}

void print_fibonacci(int n) {

   #pragma omp parallel for

   for (int i = 0; i < n; i++) {

      #pragma omp critical

      {

         printf("Thread %d: %d\n", omp_get_thread_num(), fibonacci[i]);

      }

   }

}

int main() {

   int n;

   printf("Enter the number of Fibonacci numbers to generate: ");

   scanf("%d", &n);
```

```
    #pragma omp parallel sections
    {
        #pragma omp section
        generate_fibonacci(n);


        #pragma omp section
        print_fibonacci(n);
    }
    return 0;
}
```

5. University awards gold medals to the student who has scored highest CGPA. Write aprogram to find the student with highest CGPAin a list of numbers using OpenMP.

```
#include <stdio.h>
#include <omp.h>
int main() {
    int n, i, max_index = 0;
    double max_cgpa = 0.0;
    printf("Enter the number of students: ");
    scanf("%d", &n);
    double cgpa[n];
    printf("Enter the CGPAs of the students:\n");
    for (i = 0; i < n; i++) {
        printf("Student %d: ", i + 1);
        scanf("%lf", &cgpa[i]);
    }
    #pragma omp parallel for
    for (i = 0; i < n; i++) {
        #pragma omp critical
        {
            if (cgpa[i] > max_cgpa) {
                max_cgpa = cgpa[i];
                max_index = i;
            }
        }
    }
    printf("The student with the highest CGPA is Student %d with a CGPA of %.2f\n",
max_index + 1, max_cgpa);

    return 0;
}
```

6. Assume you have n robots which pick mangoes in a farm. Write a program to calculate the total number of mangoes picked by n robots parallelly using MPI.

```
#include <stdio.h>
#include<mpi.h>
int main(int argc,char** argv) {
  int rank,size,mang,total;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  mang=rank+1;
  MPI_Reduce(&mang,&total,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
  if(rank==0){
     printf("Total mangoes picked by %d robots are %d",size,total);
  }
  MPI_Finalize();
```

```
    return 0;
}
```

7. Design a program that implements application of MPI Collective Communications.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char* argv[])
{
        int size, rank;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        float recvbuf, sendbuf[100];
        if (rank == 0) {
                int i;
                printf("Before Scatter : sendbuf of rank 0 : ");
                for (i = 0; i < size; i++) {
                        srand(i);
                        sendbuf[i] = (float)(rand()%1000)/10;
                        printf("%.1f ", sendbuf[i]);
                }
                printf("\nAfter Scatter :\n");
        }
        MPI_Scatter(sendbuf, 1, MPI_FLOAT, &recvbuf, 1, MPI_FLOAT, 0,
MPI_COMM_WORLD);
        printf("rank= %d Recvbuf: %.1f\n", rank, recvbuf);
        MPI_Finalize();
}
```

8. Implement Cartesian Virtual Topology in MPI.

```
#include <stdio.h>

#include <mpi.h>


int main(int argc, char* argv[]) {

   int rank, size, dims[2] = {0, 0}, periods[2] = {1, 1}, coords[2];

   MPI_Comm cart_comm;


   MPI_Init(&argc, &argv);

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   MPI_Comm_size(MPI_COMM_WORLD, &size);


   MPI_Dims_create(size, 2, dims);

   MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 1, &cart_comm);

   MPI_Cart_coords(cart_comm, rank, 2, coords);


   printf("Rank %d coordinates: (%d, %d)\n", rank, coords[0], coords[1]);


   MPI_Finalize();

   return 0;

}
```

9. Design a MPI program that uses blocking send/receive routines and nonblockingsend/receive routines.

```
#include <stdio.h>

#include <mpi.h>
```

```
int main(int argc, char* argv[]) {
    int rank, size, data = 0;
    MPI_Request req;
    MPI_Status status;
    double start_time, end_time, block_time, nonblock_time;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (rank == 0) {
        data = 100;

        start_time = MPI_Wtime();
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        end_time = MPI_Wtime();
        block_time = end_time - start_time;

        start_time = MPI_Wtime();
        MPI_Isend(&data, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &status);
        end_time = MPI_Wtime();
        nonblock_time = end_time - start_time;

        printf("Rank %d: Blocking time = %f, Non-blocking time = %f\n", rank, block_time,
nonblock_time);
    }
    else if (rank == 1) {
        start_time = MPI_Wtime();
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        end_time = MPI_Wtime();
        block_time = end_time - start_time;

        start_time = MPI_Wtime();
        MPI_Irecv(&data, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &status);
        end_time = MPI_Wtime();
        nonblock_time = end_time - start_time;

        printf("Rank %d: Blocking time = %f, Non-blocking time = %f\n", rank, block_time,
nonblock_time);
    }

    MPI_Finalize();
    return 0;
}
```

10. Multiply two square matrices (1000,2000 or 3000dimensions). Compare the performance of a sequential and parallel algorithm using open MP.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
int main(){
    int n;
    printf("enter n:");
    scanf("%d",&n);
```

```
int** a1=(int*)malloc(n*sizeof(int));
 int** a2=(int*)malloc(n*sizeof(int));
  int** res=(int*)malloc(n*sizeof(int));
  for(int i=0;i<n;i++){
      a1[i]=(int*)malloc(n*sizeof(int));
       a2[i]=(int*)malloc(n*sizeof(int));
        res[i]=(int*)malloc(n*sizeof(int));
  }
  omp_set_num_threads(8);
  #pragma omp parallel for
  for(int i=0;i<n;i++){
      srand(i);
      for(int j=0;j<n;j++){
          a1[i][j]=rand()%100;
          a2[i][j]=rand()%100;
      }
  }
  time_t st,et;
  st=clock();
  #pragma omp parallel for
  for(int i=0;i<n;i++){
      for(int j=0;j<n;j++){
          res[i][j]=0;
          for(int k=0;k<n;k++){
          res[i][j]+=a1[i][k]*a2[k][j];
      }
  }
  }
```

```
    et=clock();
    printf("parallel time is %lf\n",(double)(et-st)/CLOCKS_PER_SEC);


    st=clock();
    for(int i=0;i<n;i++){
      for(int j=0;j<n;j++){
        res[i][j]=0;
        for(int k=0;k<n;k++){
        res[i][j]+=a1[i][k]*a2[k][j];
      }
    }
    }
    et=clock();
    printf("sequential time is %lf\n",(double)(et-st)/CLOCKS_PER_SEC);
}
```