Roll No. 202CD005 **Sub: Introduction to Scalable Systems**
Name: Gavali Deshabhakt Nagnath

MPI Programming Assignment – 2

INDEX

Sr. No.		Page No.
1	Write an MPI program to find maximum value in an array of 600 integers with 6 processes.	2
2	Write a Parallel program to compute the prefix sums for a large sequence X[1..N] of integers.	5
3	Implement Sieve of Eratosthenes for finding list of prime numbers up to a given list.	9
4	Write a Parallel program to find minimum and maximum element in a given array of large size N.	13
5	Q5.Write parallel programs to implement Merge sort and Quick sort a) Merge Sort b) Quick Sort	18 22
6	Design a parallel scheme for computing matrix multiplication, a) Blocking Point-to-point communication b) Non Blocking Point-to-point communication c) Collective Communication	26 29 33
7	Write a MPI program to compute π MPI _Bcast and MPI _Reduce. Compare execution time for Serial code and Parallel code.	
8	Write a MPI program to compute Dot Product. a) Each process gets an equal sized chunk of both the arrays (using MPI_Scatter). b) Each process gets an unequal sized chunk of both the arrays (using MPI_Scatterv).	39 41

Q1. Write an MPI program to find maximum value in an array of 600 integers with 6 processes and print the result in root process using MPI_Reduce call. Compute time taken by the program using MPI_Wtime() function.

Program:

```
#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>

void printArray(int arr[], int r,int myrank)
{
    printf("Array at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argc, char *argv[])
{
    int rank, size,lmax,gmax;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int N=atoi(argv[1]);
    int newsize,scattersize;
    if((N%size)==0)
    {
        scattersize=(N/size);
    }
    else
    {
        scattersize=(N/size)+1;
    }
    newsize=scattersize*size;
    int ArrIN[newsize];
    for (int i = 0; i < newsize; i++)
    {
        if(i<N)
        {
            ArrIN[i] = rand()%(newsize*2);
        }
        else
        {
            ArrIN[i]=0;
        }
    }
    int B[scattersize];

MPI_Scatter(&ArrIN,scattersize,MPI_INT,&B,scattersize,MPI_INT,0,MPI_COMM_
WORLD);
    // printArray(temp,scattersize,myrank);
    lmax=B[0];
    double t1=MPI_Wtime();
    for(int i=0;i<scattersize;i++)
    {
        if(lmax<B[i])
        {
            lmax=B[i];
        }
    }
    double t2=MPI_Wtime();
    printf("Local max of process %d is %d\n",rank,lmax);
    MPI_Reduce(&lmax,&gmax,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);
    double t3=MPI_Wtime();
    printf("Time required for process %d to find loacal max is %1.10f \n",rank,(t2-t1));
    if(rank==0)
    {
        // printArray(temp,newsize,myrank);
        printf("\n\nMax of Entire array is %d\n\n",gmax);
        printf("Time required for process %d to calculate global max %1.10f \n",rank,(t3-
t2));fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

Output (with 6-Processes and Array size (N) = 600):

```
Local max of process 1 is 1195
Time required for process 1 to find loacal max is 0.0000006950
Local max of process 2 is 1190
Time required for process 2 to find loacal max is 0.0000006470
Local max of process 3 is 1187
Time required for process 3 to find loacal max is 0.0000005930
Local max of process 4 is 1189
```

```
Time required for process 4 to find loacal max is 0.0000005440
Local max of process 0 is 1182
Local max of process 5 is 1167
Time required for process 0 to find loacal max is 0.0000005380


Max of Entire array is 1195

Time required for process 0 to calculate global max 0.0002270930
Time required for process 5 to find loacal max is 0.0000005300
```

Q2. Write a Parallel program to compute the prefix sums for a large sequence X[1..N] of integers. Take large enough values of N.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>

void printArray(int arr[], int r,int myrank)
{
    printf("Array at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argc,char* argv[])
{
    int size,myrank;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    int N=atoi(argv[1]);    // Taking input from command line
    int scattersize;
    if((N%size)==0)
    {
        scattersize=(N/size);
    }
    else
    {
        scattersize=(N/size)+1;
    }
    int newsize=scattersize*size;
    int a[newsize];
    if(myrank==0)
    {
        for(int i=0;i<newsize;i++)
        {
            if(i<N)
            {
                a[i]=i+1;
            }
            else
            {
                a[i]=0;
            }

        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    int temp[scattersize];

MPI_Scatter(&a,scattersize,MPI_INT,&temp,scattersize,MPI_INT,0,MPI_COMM_W
ORLD);
    double t1=MPI_Wtime();
    for(int i=0;i<scattersize;i++)
    {
        if(i!=0)
        {
            int t=temp[i];
            temp[i]=t+temp[i-1];
        }
    }
    double t2=MPI_Wtime();
    printf("\nProcesse: %d --> Time required  to perform PreFix Sum: %1.10f\
n",myrank,(t2-t1));
    printArray(temp,scattersize,myrank);
    int gArray[newsize];

MPI_Gather(&temp,scattersize,MPI_INT,&gArray,scattersize,MPI_INT,0,MPI_COM
M_WORLD);
    if(myrank==0 && size>1)
    {
        double t3=MPI_Wtime();
        for(int i=scattersize;i<N;i++)
        {
            {
                int t=gArray[i];
                gArray[i]=t+gArray[i-1];
            }
        }
        double t4=MPI_Wtime();
        printf("\nProcesse: %d --> Time required  to perform Final PreFix Sum: %1.10f\
n",myrank,(t3-t2));
        printArray(gArray,newsize,myrank);
    }
    MPI_Finalize();
    return 0;
}
```

Output:
I) With 1-Process (Serial) and N 50

```
Processe: 0 --> Time required  to perform PreFix Sum: 0.0000003940
Array at process 0 is 1  3  6  10  15  21  28  36  45  55  66  78  91  105  120  136  153
171  190  210  231  253  276  300  325  351  378  406  435  465  496  528  561  595
630  666  703  741  780  820  861  903  946  990  1035  1081  1128  1176  1225  1275
```

II) With 2-Processes (Parallel) and N 50

```
Processe: 0 --> Time required  to perform PreFix Sum: 0.0000002870
Array at process 0 is 1  3  6  10  15  21  28  36  45  55  66  78  91  105  120  136  153
171  190  210  231  253  276  300  325

Processe: 0 --> Time required  to perform Final PreFix Sum: 0.0000390200

Processe: 1 --> Time required  to perform PreFix Sum: 0.0000002480
Array at process 1 is 26  53  81  110  140  171  203  236  270  305  341  378  416  455
495  536  578  621  665  710  756  803  851  900  950
Array at process 0 is 1  3  6  10  15  21  28  36  45  55  66  78  91  105  120  136  153
171  190  210  231  253  276  300  325  351  404  485  595  735  906  1109  1345
1615  1920  2261  2639  3055  3510  4005  4541  5119  5740  6405  7115  7871  8674
9525  10425  11375
```

Table 2.1: PreFix Sum Serial Run

Sr. No.	N	time
1.	50	0.0000004210
2.	100	0.0000006090
3.	500	0.0000023470
4.	5000	0.0000310410
5.	50000	0.0003103050
6.	500000	0.0021541310

Table 2.2: PreFix Sum Parallel Run (Time)

n\N	50	100	500	5000	50000	500000
2	2.7384e-05	2.9513e-05	3.2224e-05	8.0137e-05	0.000372099	0.002637091
4	0.000128466	0.000120956	0.000168512	0.000244656	0.000964675	0.010348513
8	0.000824933	0.001297961	0.000999324	0.000449815	0.018602575	0.014343954
12	0.001237421	0.001298486	0.001195486	0.000549916	0.019715551	0.023031060
18	0.002558431	0.002326414	0.013810833	0.004803454	0.024259127	0.031744568
24	0.004202694	0.021684437	0.023117531	0.010309192	0.026595193	0.033136005

Conclusion:
I) For a given number of processes, as N increases, the run time also goes on increasing.
II) For a given N, as number of processes increases, run time also goes on increasing.

Q3. Implement Sieve of Eratosthenes for finding list of prime numbers up to a given list.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

void printArray(int arr[], int r,int myrank)
{
    printf("Array at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argc, char *argv[])
{
    int myrank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    int N = atoi(argv[1]) -1;               //Taking input from Command line
    int scattersize;
    if ((N - 1) % size == 0)
    {
        scattersize = (N - 1) / size;
    }
    else
    {
        scattersize = ((N - 1) / size) + 1;
    }
    int newsize = (size * scattersize);
    int a[newsize], b[scattersize], c[newsize];

    if (myrank == 0)
    {
        for (int i = 0; i < newsize; i++)
        {
            if (i < (N - 1))
            {
                a[i] = i + 2;
            }
            else
            {
                a[i] = 0;
            }
        }
    }
    MPI_Scatter(&a, scattersize, MPI_INT, &b, scattersize, MPI_INT, 0,
MPI_COMM_WORLD);
    double t1 = MPI_Wtime();
    int i = 0, x;
    while (i < scattersize)
    {
        x = b[i] - 1;
        while (x >= 2)
        {
            if ((b[i] % x) == 0 && b[i] != 2)
            {
                b[i] = 0;
                break;
            }
            x--;
        }
        i++;
    }
    double t2 = MPI_Wtime();

    // printArray(temp,scattersize,myrank);

printf("\nProcess: %d -> Time Required to find Prime's : %1.10f \n",myrank,(t2-t1));

MPI_Gather(&b, scattersize, MPI_INT, &c, scattersize, MPI_INT, 0,

MPI_COMM_WORLD);
    if (myrank == 0)
    {
        printf("\nThe prime numbers are as follows:\n");

        for (int i = 0; i < newsize; i++)
        {
            if (c[i] != 0)
            {
                printf("%d ", c[i]);
            }
        }
    }

    MPI_Finalize();
    return 0;
}
```

Output:

1. Processes = 1 (Serial) and N = 50

```
Process: 0 -> Time Required to find Prime's : 0.0000053120

The prime no's are as follows:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

2. Processes = 2 (Parallel) and N = 50

```
Process: 0 -> Time Required to find Prime's : 0.0000017980

The prime no's are as follows:

Process: 1 -> Time Required to find Prime's : 0.0000035180
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Table 3.1 : Sieve of Eratosthenes – Serial execution (i.e. n=1)

Sr. No.	N	Time
1	50	0. 0000053120
2	100	0.0000147130
3	200	0.0000590170
4	500	0.0003868790
5	1000	0.0017747070

Table 3.2 : Sieve of Eratosthenes – Parallel execution (time)

n\N	50	100	200	500	1000
2	0.0000035340	0.0000123350	0.0001463700	0.0004041340	0.0019982600
4	0.0000029980	0.0000089470	0.0001630450	0.0003174730	0.0014760660
5	0.0000022840	0.0000094530	0.0001216030	0.0001196420	0.0011302580
10	0.0000011860	0.0000038540	0.0001096700	0.0001021880	0.0005455280

Conclusion:
I) For a given Array size (or Upper bound), As number of Processes incresases, execution time goes on decreasing.
II) For a given number of processes, As Array size increases, execution time goes on increasing.

Q4. Write a Parallel program to find minimum and maximum element in a given array of large size N.

Program:

```
// Find Min and Max from an Array of size N

#include<stdio.h>
#include<mpi.h>
#include <stdlib.h>
#include<time.h>
void printArray(int arr[], int r,int myrank)
{
    printf("Array at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}

int main(int argc,char* argv[])
{
    int size,myrank,gmin,gmax;
    srand(time(0));
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    int N=atoi(argv[1]);                //atoi -->Used to convert char to int &&
                                        // atoi(argv[1])-->used to take arraysize input from command
line
    int scattersize,newsize;
    if((N%size)==0)
    {
        scattersize=(N/size);
    }
    else
    {
        scattersize=(N/size)+1;
    }
    newsize=scattersize*size;
    int a[newsize];
    if(myrank==0)
    {
        for(int i=0;i<newsize;i++)
        {
            if(i<N)
            {
                a[i]=rand()%(newsize*(2));
            }
            else
            {
                a[i]=0;
            }

        }
    }
    int localArray[scattersize];

MPI_Scatter(&a,scattersize,MPI_INT,&localArray,scattersize,MPI_INT,0,MPI_COM
M_WORLD);
    double t1=MPI_Wtime();
    int lmax=localArray[0];
    for(int i=0;i<scattersize;i++)
    {
        if(lmax<localArray[i])
        {
            lmax=localArray[i];
        }
    }
    printf("\n\nMAX of array at process %d is %d\n\n",myrank,lmax);
    double t2=MPI_Wtime();
    printf("Time required for process %d to calculate local MAX %1.10f \n",myrank,
(t2-t1));
    printArray(localArray,scattersize,myrank);
    int lmin=localArray[0];
    for(int i=0;i<scattersize;i++)
    {

        if(lmin>localArray[i]&&localArray[i]!=0)
        {
            lmin=localArray[i];
        }
    }
    printf("\n\nMIN of array at process %d is %d\n\n",myrank,lmin);
    double t3=MPI_Wtime();
    printf("Time required for process %d to calculate local MIN %1.10f \n",myrank,(t3-
t2));
    MPI_Reduce(&lmin,&gmin,1,MPI_INT,MPI_MIN,0,MPI_COMM_WORLD);
    double t4=MPI_Wtime();
    MPI_Reduce(&lmax,&gmax,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);
    double t5=MPI_Wtime();
    if(myrank==0 && size>1)
    {
        printArray(a,newsize,myrank);
        printf("\n\nMIN of Entire array is %d\n\n",gmin);
        printf("\n\nMAX of Entire array is %d\n\n",gmax);
        printf("Time required for process %d to calculate global MIN %1.10f \n",myrank,
(t4-t3));
        printf("Time required for process %d to calculate global MAX %1.10f \
n",myrank,(t5-t4));
    }
    MPI_Finalize();
    return 0;
}
```

Output:
I) Processes = 1 (Serial) and N 50

```
MAX of array at process 0 is 97

Time required for process 0 to calculate local MAX 0.0000768500
Array at process 0 is 97  57  79  82  22  88  40  52  94  41  78  15  60  94  84  60  38
39  50  35  85  44  38  8  38  4  52  28  37  1  23  35  59  3  69  33  91  61  85  38  2  15
53  15  9  38  27  47  29  78


MIN of array at process 0 is 1

Time required for process 0 to calculate local MIN 0.0000413650
```

II) Processes = 2 (Parallel) and N 50

```
MAX of array at process 0 is 96

Time required for process 0 to calculate local MAX 0.0000345900
```

```
Array at process 0 is 88  10  14  34  36  25  17  95  49  91  6  66  74  20  31  24  85  33
59  44  54  13  14  6  96


MIN of array at process 0 is 6

Time required for process 0 to calculate local MIN 0.0000274570


MAX of array at process 1 is 94

Time required for process 1 to calculate local MAX 0.0000438090
Array at process 1 is 88  2  29  79  83  27  19  45  94  53  33  19  70  28  68  14  87  86
88  7  69  64  45  3  23


MIN of array at process 1 is 2

Time required for process 1 to calculate local MIN 0.0000295770
Array at process 0 is 88  10  14  34  36  25  17  95  49  91  6  66  74  20  31  24  85  33
59  44  54  13  14  6  96  88  2  29  79  83  27  19  45  94  53  33  19  70  28  68  14  87
86  88  7  69  64  45  3  23


MIN of Entire array is 2



MAX of Entire array is 96

Time required for process 0 to calculate global MIN 0.0000512110
Time required for process 0 to calculate global MAX 0.0000016990
```

Table 4.1 : Minimum and Maximum from Array – Serial execution (i.e. n=1)

Sr. No.	N	Time for Finding Min	Time for Finding Max
1	500	0.0000017190	0.0000260900
2	1000	0.0000033250	0.0000250680
3	2000	0.0000066820	0.0000314880
4	5000	0.0000211590	0.0000485550
5	10000	0.0000322740	0.0000558640

Table 4.2 : Time required to find minimum from Array – Parallel execution (time)

n\N	500	1000	2000	5000	10000
2	0.0000462670	0.0000822590	0.0001552880	0.0000681570	0.0000709850
4	0.0000029980	0.0005919010	0.0003339740	0.0001224330	0.0000626950
8	0.0000022840	0.0057549470	0.0005551600	0.0002621310	0.0001302580
12	0.0000011860	0.0099092960	0.0007481430	0.0004021880	0.0005455280

Table 4.3 : Time required to find Maximum from Array – Parallel execution (time)

n\N	500	1000	2000	5000	10000
2	0.0000425380	0.0000437490	0.0000452880	0.0000561570	0.0000589850
4	0.0000029980	0.0005919010	0.0003339740	0.0001224330	0.0004726950
8	0.0000022840	0.0057549470	0.0005551600	0.0002621310	0.0033083920
12	0.0000011860	0.0099092960	0.0007481430	0.0032148520	0.0118603990

Conclusion:
I) For a given number of processes, As N increases, run time goes on increasing in both cases i.e. while finding maximum and minimum from array.
II) For a given N, As number of processes increases, run time goes on decreasing in both cases i.e. while finding maximum and minimum from array.

Q5. Write parallel programs to implement Merge sort and Quick sort

a) Merge Sort
Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
void merge(int arr[],int l,int m, int r)
{
    int i=l,j=m+1,k=l;
    int temp[r+1];
    while(i<=m&&j<=r)
    {
        if(arr[i]<arr[j])
        {
            temp[k]=arr[i];
            i++;k++;
        }
        else
        {
            temp[k]=arr[j];
            j++;k++;
        }
    }
    while(i<=m)
    {
        temp[k]=arr[i];
        i++;k++;
    }
    while(j<=r)
    {
        temp[k]=arr[j];
        j++;k++;
    }
    for(int s=l;s<=r;s++)
    {
        arr[s]=temp[s];
    }
}
void mergeSort(int arr[],int l,int r)
{
    if(l<r)
    {
        int m=(l+r)/2;
        mergeSort(arr,l,m);
        mergeSort(arr,m+1,r);
        merge(arr,l,m,r);
    }
}
void printArray(int arr[],int r)
{
    for(int i=0;i<=r;i++)
    {
        printf("%d  ",arr[i]);
    }
    printf("\n");
}
int main(int argc,char* argv[])
{
    int size,myrank;
    int N=atoi(argv[1]);  // tanking input from command line
    int array[N];
    for (int i = 0; i < N; i++)
    {
        array[i] = rand() % (N * 2);
    }
    int r=N-1;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    int localArray[N/size];

MPI_Scatter(&array,(N/size),MPI_INT,&localArray,(N/size),MPI_INT,0,MPI_COM
M_WORLD);
    double t1=MPI_Wtime();
    mergeSort(localArray,0,(r/size));
    double t2=MPI_Wtime();
    printf("\nSorted LOCAL ARRAY AT PROCESS %d is ",myrank);
    printArray(localArray,(r/size));
    printf("\nTime required for Process %d to Sort local array is %1.10f \n",myrank,(t2-
t1));

MPI_Gather(&localArray,(N/size),MPI_INT,&array,(N/size),MPI_INT,0,MPI_COM
M_WORLD);
    if(myrank==0 && size >1)
    {
       double t3=MPI_Wtime();
       if(size!=1)
         mergeSort(array,0,r);
       double t4=MPI_Wtime();
       printf("\n\nSORTED GLOBAL ARRAY AT PROCESS %d is ",myrank);
       printArray(array,r);
       printf("\nTIME REQUIRED FOR PROCESS %d TO SORT LOCAL ARRAYS
(RECEIVED FORM MPI_Gather()) is %1.10f \n",myrank, (t4-t3));
    }
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

Output:

I) With 1-process (Serial) and N 200

```
Sorted LOCAL ARRAY AT PROCESS 0 is 8  11  12  13  14  19  21  22  27  28  28  29
29  29  31  37  37  40  42  43  44  46  46  56  58  59  62  62  67  71  75  76  76  81  81
86  89  91  91  92  92  94  97  97  101  102  105  106  108  113  114  115  117  117  119
121  121  124  124  125  127  129  132  136  137  139  140  143  143  145  149  150
151  163  164  164  165  167  167  167  167  168  168  168  170  171  173  180  182
183  184  186  186  186  187  188  193  193  193  196  196  201  202  203  207  211
211  218  218  219  219  221  224  226  226  227  228  229  232  234  234  234  235
241  241  245  249  251  251  252  256  256  257  258  259  260  265  267  269  270
273  274  275  280  283  284  286  287  288  290  295  297  300  303  305  309  315
315  323  323  324  326  326  327  328  328  329  329  329  330  332  335  335  336
336  339  340  350  353  354  355  356  356  362  364  365  370  370  372  377  378
378  379  382  384  393  395  395  398  399

Time required for Process 0 to Sort local array is 0.0000507130
```

II) With 2-processes (Parallel) and N 200

```
Sorted LOCAL ARRAY AT PROCESS 0 is 8  11  12  13  14  21  22  27  29  42  43  46
56  59  62  62  67  76  76  81  86  91  92  94  105  113  115  119  124  125  129  136
137  139  140  145  150  163  167  167  167  168  168  173  180  182  183  184  186
186  188  193  193  196  202  203  211  221  226  226  229  234  249  251  257  258
```

```
260  269  273  284  287  290  295  305  315  323  324  326  326  327  329  330  332
335  335  336  339  354  356  362  364  370  370  372  377  378  382  384  398  399

Time required for Process 0 to Sort local array is 0.0000135650

Sorted LOCAL ARRAY AT PROCESS 1 is 19  28  28  29  29  31  37  37  40  44  46
58  71  75  81  89  91  92  97  97  101  102  106  108  114  117  117  121  121  124
127  132  143  143  149  151  164  164  165  167  168  170  171  186  187  193  196
201  207  211  218  218  219  219  224  227  228  232  234  234  235  241  241  245
251  252  256  256  259  265  267  270  274  275  280  283  286  288  297  300  303
309  315  323  328  328  329  329  336  340  350  353  355  356  365  378  379  393
395  395

Time required for Process 1 to Sort local array is 0.0000137230


SORTED GLOBAL ARRAY AT PROCESS 0 is 8  11  12  13  14  19  21  22  27  28
28  29  29  29  31  37  37  40  42  43  44  46  46  56  58  59  62  62  67  71  75  76  76
81  81  86  89  91  91  92  92  94  97  97  101  102  105  106  108  113  114  115  117
117  119  121  121  124  124  125  127  129  132  136  137  139  140  143  143  145
149  150  151  163  164  164  165  167  167  167  167  168  168  168  170  171  173
180  182  183  184  186  186  186  187  188  193  193  193  196  196  201  202  203
207  211  211  218  218  219  219  221  224  226  226  227  228  229  232  234  234
234  235  241  241  245  249  251  251  252  256  256  257  258  259  260  265  267
269  270  273  274  275  280  283  284  286  287  288  290  295  297  300  303  305
309  315  315  323  323  324  326  326  327  328  328  329  329  329  330  332  335
335  336  336  339  340  350  353  354  355  356  356  362  364  365  370  370  372
377  378  378  379  382  384  393  395  395  398  399

TIME REQUIRED FOR PROCESS 0 TO SORT LOCAL ARRAYS (RECEIVED
FORM MPI_Gather()) is 0.0000221850
```

Table 5.1 : Serial Execution time of Merge Sort for different Array size

Sr. No.	N	Time
1	200	0.0000307970
2	500	0.0001386160
3	1000	0.0001799850
4	5000	0.0009949870

Table 5.2 : Parallel Execution time of Merge Sort for different Array size (N) and no. of processes (n)

n\N	200	500	1000	5000
2	0.0000213580	0.0000932520	0.0001086330	0.0008472210
4	0.0000275940	0.0000908220	0.0001907010	0.0011158780
5	0.0000306420	0.0000871810	0.0002210400	0.0011457550
10	0.0000414420	0.0000840050	0.0002425140	0.0014612690

b) Quick Sort
Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int Partition(int arr[],int s,int e)
{
    int pivot=arr[e];
    int pIndex=s;
    for(int i=s;i<e;i++)
    {
        if(arr[i]<pivot)
        {
            int temp = arr[i];
            arr[i]=arr[pIndex];
            arr[pIndex]=temp;
            pIndex++;
        }
    }
    int temp = arr[e];
    arr[e] = arr[pIndex];
    arr[pIndex] = temp;
    return pIndex;
}
void QuickSort(int arr[],int s,int e)
{
    if(s<e)
    {
        int p = Partition(arr,s,e);
        QuickSort(arr,s,p-1);
        QuickSort(arr,p+1,e);
    }
```

```
}
void printArray(int arr[], int r)
{
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argc, char *argv[])
{
    int size, myrank;
    int N=atoi(argv[1]);  // tanking input from command line
    int array[N];
    for (int i = 0; i < N; i++)
    {
        array[i] = rand() % (N * 2);
    }
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    int localArray[N / size];
    MPI_Scatter(&array, (N / size), MPI_INT, &localArray, (N / size), MPI_INT, 0,
MPI_COMM_WORLD);
    double t1 = MPI_Wtime();
    QuickSort(localArray, 0, (N / size));
    double t2 = MPI_Wtime();
    printf("\nSorted LOCAL ARRAY AT PROCESS %d is ",myrank);
    printArray(localArray, (N / size));
    printf("\nTime required for Process %d to Sort local array is %1.10f \n", myrank, (t2
- t1));
    MPI_Gather(&localArray, (N / size), MPI_INT, &array, (N / size), MPI_INT, 0,
MPI_COMM_WORLD);
    if (myrank == 0 && size >1)
    {
        double t3 = MPI_Wtime();
        if (size != 1)
            QuickSort(array, 0, N);
        double t4 = MPI_Wtime();
        printf("\n\nSORTED GLOBAL ARRAY AT PROCESS %d is ",myrank);
        printArray(array, N);
        printf("\nTIME REQUIRED FOR PROCESS %d TO SORT LOCAL ARRAYS
(RECEIVED FORM MPI_Gather()) is %1.10f \n", myrank, (t4 - t3));
```

```
    }
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

Output:

I) With 1-process (Serial) and N 200

```
Sorted LOCAL ARRAY AT PROCESS 0 is 8  11  12  13  14  19  21  22  27  28  28  29
29  29  31  37  37  40  42  43  44  46  46  56  58  59  62  62  67  71  75  76  76  81  81
86  89  91  91  92  92  94  97  97  101  102  105  106  108  113  114  115  117  117  119
121  121  124  124  125  127  129  132  136  137  139  140  143  143  145  149  150
151  163  164  164  165  167  167  167  167  168  168  168  170  171  173  180  182
183  183  184  186  186  186  187  188  193  193  193  196  196  201  202  203  207
211  211  218  218  219  219  221  224  226  226  227  228  229  232  234  234  234
235  241  241  245  249  251  251  252  256  256  257  258  259  260  265  267  269
270  273  274  275  280  283  284  286  287  288  290  295  297  300  303  305  309
315  315  323  323  324  326  326  327  328  328  329  329  329  330  332  335  335
336  336  339  340  350  353  354  355  356  356  362  364  365  370  370  372  377
378  378  379  382  384  393  395  395  398

Time required for Process 0 to Sort local array is 0.0000215430
```

II) With 2-processes (Parallel) and N 200

```
Sorted LOCAL ARRAY AT PROCESS 0 is 8  11  12  13  14  21  22  27  29  42  43  46
56  59  62  62  67  76  76  81  86  91  92  94  105  113  115  119  124  125  129  136
137  139  140  145  150  163  167  167  167  168  168  173  180  182  183  183  184
186  186  188  193  193  196  202  203  211  221  226  226  229  234  249  251  257
258  260  269  273  284  287  290  295  305  315  323  324  326  326  327  329  330
332  335  335  336  339  354  356  362  364  370  370  372  377  378  382  384  398

Time required for Process 0 to Sort local array is 0.0000095780


SORTED GLOBAL ARRAY AT PROCESS 0 is

Sorted LOCAL ARRAY AT PROCESS 1 is 19  28  28  29  29  31  37  37  40  44  46
58  71  75  81  89  91  92  97  97  101  102  106  108  114  117  117  121  121  124
127  132  143  143  149  151  164  164  165  167  168  170  171  183  186  187  193
196  201  207  211  218  218  219  219  224  227  228  232  234  234  235  241  241
245  251  252  256  256  259  265  267  270  274  275  280  283  286  288  297  300
```

```
303  309  315  323  328  328  329  329  336  340  350  353  355  356  365  378  379
393  395

Time required for Process 1 to Sort local array is 0.0000092920
8  11  12  13  14  19  21  22  27  28  28  29  29  29  31  37  37  40  42  43  44  46  46
56  58  59  62  62  67  71  75  76  76  81  81  86  89  91  91  92  92  94  97  97  100
101  102  105  106  108  113  114  115  117  117  119  121  121  124  124  125  127
129  132  136  137  139  140  143  143  145  149  150  151  163  164  164  165  167
167  167  167  168  168  168  170  171  173  180  182  183  183  183  184  186  186
186  187  188  193  193  193  196  196  201  202  203  207  211  211  218  218  219
219  221  224  226  226  227  228  229  232  234  234  234  235  241  241  245  249
251  251  252  256  256  257  258  259  260  265  267  269  270  273  274  275  280
283  284  286  287  288  290  295  297  300  303  305  309  315  315  323  323  324
326  326  327  328  328  329  329  329  330  332  335  335  336  336  339  340  350
353  354  355  356  356  362  364  365  370  370  372  377  378  378  379  382  384
393  395

TIME REQUIRED FOR PROCESS 0 TO SORT LOCAL ARRAYS (RECEIVED
FORM MPI_Gather()) is 0.0000252780
```

Table 5.3 : Serial Execution time of Quick Sort for different Array size

Sr. No.	N	Time
1	200	0.0000263250
2	500	0.0000723720
3	1000	0.0001283400
4	5000	0.0007881160

Table 5.4 : Parallel Execution time of Quick Sort for different Array size (N) and no. of processes (n)

n\N	200	500	1000	5000
2	0.0000392640	0.0002264200	0.0006234560	0.0195855210
4	0.0000336550	0.0001534630	0.0004707010	0.0100585860
5	0.0000292760	0.0001328400	0.0003010400	0.0094874490
10	0.0000259880	0.0000857200	0.0002486880	0.0052751830

Conclusion:
I) Quick sort algorithm gives faster results on small size array.
II) Merge sort algorithm gives faster results on large size array.

Q6. Write MPI program to calculate the product of two matrices A (of size N*32) and B (of size 32*N), which should be a N*N matrix. Design a parallel scheme for computing matrix multiplication,

a) Blocking P2P (point-to-point) communication

Program:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
// #define N 3                /* number of rows in matrix A and number of columns in
matrix B*/
#define FROM_MASTER 1         /* setting a message type */
#define FROM_WORKER 2         /* setting a message type */

int main (int argc, char *argv[])
{
int N=atoi(argv[1]);     /* Taking input from command line*/
int       size,              /* number of processes in partition */
          myrank,               /* a task identifier */
          numworkers,            /* number of worker processes */
          source,               /* task id of message source */
          dest,                 /* task id of message destination */
          mtype,                /* message type */
          rows,                 /* rows of matrix A sent to each worker */
          averow, extra, offset, /* used to determine rows sent to each worker */
          i, j, k, rc;          /* misc */
double a[N][32],           /* matrix A to be multiplied */
          b[32][N],          /* matrix B to be multiplied */
          c[N][N];           /* result matrix C */
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
MPI_Comm_size(MPI_COMM_WORLD,&size);
if (size < 2 ) {
  printf("Need at least two MPI processes. Quitting...\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  exit(1);
  }
numworkers = size-1;
```

```
/*************************** master task
************************************/
  if (myrank == 0)
  {
    printf("Initializing arrays...\n");
    for (i=0; i<N; i++)
      for (j=0; j<32; j++)
        a[i][j]= i+j;
    for (i=0; i<32; i++)
      for (j=0; j<N; j++)
        b[i][j]= i-j;

    /* Send matrix data to the worker processes */
    averow = N/numworkers;
    extra = N%numworkers;
    offset = 0;
    mtype = FROM_MASTER;
    for (dest=1; dest<=numworkers; dest++)
    {
      rows = (dest <= extra) ? averow+1 : averow;
      MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
      MPI_Send(&a[offset][0], rows*32, MPI_DOUBLE, dest, mtype,
             MPI_COMM_WORLD);
      MPI_Send(&b, 32*N, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);
      offset = offset + rows;
    }

    /* Receive results from worker processes */
    mtype = FROM_WORKER;
    for (i=1; i<=numworkers; i++)
    {
      source = i;
      MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
&status);
      MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
&status);
      MPI_Recv(&c[offset][0], rows*N, MPI_DOUBLE, source, mtype,
            MPI_COMM_WORLD, &status);
      printf("Received results from task %d\n",source);
    }

    /* Print results */
    printf("******************************************************\n");
    printf("Result Matrix:\n");
    for (i=0; i<N; i++)
    {
      printf("\n");
      for (j=0; j<N; j++)
        printf("%8.2f   ", c[i][j]);
    }
    printf("\n******************************************************\n");
    printf ("Done.\n");
  }


/**************************** worker task
************************************/
  if (myrank > 0)
  {
    mtype = FROM_MASTER;
    double t1 = MPI_Wtime();
    MPI_Recv(&offset, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD, &status);
    MPI_Recv(&rows, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD, &status);
    MPI_Recv(&a, rows*32, MPI_DOUBLE, 0, mtype, MPI_COMM_WORLD,
&status);
    MPI_Recv(&b, 32*N, MPI_DOUBLE, 0, mtype, MPI_COMM_WORLD,
&status);
    for (k=0; k<N; k++)
      for (i=0; i<rows; i++)
      {
        c[i][k] = 0.0;
        for (j=0; j<32; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
    double t2 = MPI_Wtime();
    printf("\nTotal Run Time: %1.10f \n",myrank,(t2-t1));
    mtype = FROM_WORKER;
    MPI_Send(&offset, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD);
    MPI_Send(&rows, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD);
    MPI_Send(&c, rows*N, MPI_DOUBLE, 0, mtype, MPI_COMM_WORLD);
  }
  MPI_Finalize();
}
```

Output:

```
Initializing arrays...

Total Run Time: 0.0002143140
Received results from task 1
******************************************************
Result Matrix:

10416.00    9920.00    9424.00    8928.00
10912.00   10384.00    9856.00    9328.00
11408.00   10848.00   10288.00    9728.00
11904.00   11312.00   10720.00   10128.00
******************************************************
Done.
```

b) Non-Blocking P2P (point-to-point) communication

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define FROM_MASTER 1            //tag for messages sent from master to slaves
#define FROM_WORKER 4            //tag for messages sent from slaves to master

int main(int argc, char *argv[])
{
  int N = atoi(argv[1]); /* Taking input from command line*/
  int myrank, size, i, j, k, rc, low_bound, upper_bound, portion;
  double a[N][32], b[32][N], c[N][N];
  double t1, t2;
  MPI_Status status;                    // store status of a MPI_Recv
  MPI_Request request;                   //capture request of a MPI_Isend
  MPI_Init(&argc, &argv);                //initialize MPI operations
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank); //get the myrank
  MPI_Comm_size(MPI_COMM_WORLD, &size);   //get number of processes
  if (size < 2)
  {
    printf("Need at least two MPI processes. Quitting...\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
    exit(1);
  }
  /* master initializes work*/
  if (myrank == 0)
  {
    for (i = 0; i < N; i++)
    {
      printf("Initializing arrays...\n");
      for (j = 0; j < 32; j++)
      {
        a[i][j] = i + j;
      }
    }
    for (i = 0; i < 32; i++)
    {
      for (j = 0; j < N; j++)
      {
        b[i][j] = i - j;
      }
    }
    t1 = MPI_Wtime();
    for (i = 1; i < size; i++)
    {
      portion = (N / (size - 1));
      low_bound = (i - 1) * portion;
      if (((i + 1) == size) && ((N % (size - 1)) != 0))
      {
        upper_bound = N;
      }
      else
      {
        upper_bound = low_bound + portion;
      }
      MPI_Isend(&low_bound, 1, MPI_INT, i, FROM_MASTER,
MPI_COMM_WORLD, &request);
      MPI_Isend(&upper_bound, 1, MPI_INT, i, FROM_MASTER + 1,
MPI_COMM_WORLD, &request);
      MPI_Isend(&a[low_bound][0], (upper_bound - low_bound) * 32,
MPI_DOUBLE, i, FROM_MASTER + 2, MPI_COMM_WORLD, &request);
    }
  }
  MPI_Bcast(&b, 32 * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

  /* work done by slaves*/
  if (myrank > 0)
  {
    MPI_Recv(&low_bound, 1, MPI_INT, 0, FROM_MASTER,
MPI_COMM_WORLD, &status);
    MPI_Recv(&upper_bound, 1, MPI_INT, 0, FROM_MASTER + 1,
MPI_COMM_WORLD, &status);
    MPI_Recv(&a[low_bound][0], (upper_bound - low_bound) * 32, MPI_DOUBLE,
0, FROM_MASTER + 2, MPI_COMM_WORLD, &status);
    for (i = low_bound; i < upper_bound; i++)
    {
      for (j = 0; j < N; j++)
      {
        for (k = 0; k < 32; k++)
        {
          c[i][j] += (a[i][k] * b[k][j]);
        }
      }
    }

    MPI_Isend(&low_bound, 1, MPI_INT, 0, FROM_WORKER,
MPI_COMM_WORLD, &request);
    MPI_Isend(&upper_bound, 1, MPI_INT, 0, FROM_WORKER + 1,
MPI_COMM_WORLD, &request);
    MPI_Isend(&c[low_bound][0], (upper_bound - low_bound) * N, MPI_DOUBLE,
0, FROM_WORKER + 2, MPI_COMM_WORLD, &request);
  }

  /* master gathers processed work*/
  if (myrank == 0)
  {
    for (i = 1; i < size; i++)
    {
      MPI_Recv(&low_bound, 1, MPI_INT, i, FROM_WORKER,
MPI_COMM_WORLD, &status);
      MPI_Recv(&upper_bound, 1, MPI_INT, i, FROM_WORKER + 1,
MPI_COMM_WORLD, &status);
      MPI_Recv(&c[low_bound][0], (upper_bound - low_bound) * N,
MPI_DOUBLE, i, FROM_WORKER + 2, MPI_COMM_WORLD, &status);
    }
    t2 = MPI_Wtime();
    printf("\nTotal Run Time: %1.10f \n", myrank, (t2 - t1));
    /* Print results */
    printf("******************************************************\n");
    printf("Result Matrix:\n");
    for (i = 0; i < N; i++)
    {
      printf("\n");
      for (j = 0; j < N; j++)
        printf("%8.2f   ", c[i][j]);
    }
    printf("\n******************************************************\n");
    printf("Done.\n");
  }
  MPI_Finalize();
  return 0;
}
```

Output:

```
Initializing arrays...

Total Run Time: 0.0000402780
******************************************************
Result Matrix:

10416.00    9920.00    9424.00    8928.00
10912.00   10384.00    9856.00    9328.00
11408.00   10848.00   10288.00    9728.00
11904.00   11312.00   10720.00   10128.00
******************************************************
Done.
```

c) Collective communication

Program:

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <stdlib.h>
#include <stddef.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
  int i, j, k, rc, myrank, size, tag = 99;
  int sum = 0;
  int N=atoi(argv[1]);     /* Taking input from command line*/

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  int from = myrank * (N*32)/size;
  int to = (myrank+1) * (N*32)/size;
  double aa[N * 32 / size], cc[N *N];
  double a[N][32],b[32][N];
  double c[N][N];
  if (size < 2)
  {
    printf("Need at least two MPI processes. Quitting...\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
    exit(1);
  }
  double t1 = MPI_Wtime();
  if(myrank==0)
  {

  printf("Initializing arrays...\n");
  for (int i = 0; i < N; i++)
  {
    for (int j = 0; j < 32; j++)
    {
      a[i][j] = i + j;
    }
  }
  for (int i = 0; i < 32; i++)
  {
    for (int j = 0; j < N; j++)
    {
      b[i][j] = i - j;
    }
  }
  }
  //scatter rows of first matrix to different processes
  MPI_Scatter(&a, N * 32/size, MPI_INT, &aa, N * 32/size, MPI_INT, 0,
MPI_COMM_WORLD);

  //broadcast second matrix to all processes
  MPI_Bcast(&b, 32 * N, MPI_INT, 0, MPI_COMM_WORLD);

  MPI_Barrier(MPI_COMM_WORLD);

  //performing vector multiplication by all processes
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < 32; j++)
    {
      sum = sum + aa[j] * b[j][i];
    }
    cc[i] = sum;
    sum = 0.00;
  }
  for (i = 0; i < N; i++)         // for each row of A
        {
    for (j = 0; j < N; j++)       // for each column of B
                {
      c[i][j] = 0;
                        for (k = 0; k < 32; k++)
                        {
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
      }
    }
  }
  MPI_Gather(cc, N * N/size, MPI_INT, c, N * N, MPI_INT, 0,
MPI_COMM_WORLD);
  double t2 = MPI_Wtime();
  printf("\nTotal Run Time of process %d : %1.10f \n", myrank, (t2 - t1));
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
  if (myrank == 0)
  {
    printf("******************************************************\n");
    printf("Result Matrix:\n");
    for (i = 0; i < N; i++)
    {
      printf("\n");
      for (j = 0; j < N; j++)
        printf("%1.2f   ", c[i][j]);
    }
    printf("\n******************************************************\n");
    printf("Done.\n");
  }
  return 0;
}
```

Output:

```
Initializing arrays...

Total Run Time of process 0 : 0.0000698900
Total Run Time of process 1 : 0.0001003040
******************************************************
Result Matrix:

10416.00    9920.00    9424.00    8928.00
10912.00   10384.00    9856.00    9328.00
11408.00   10848.00   10288.00    9728.00
11904.00   11312.00   10720.00   10128.00
******************************************************
Done.
```

Q7. Write a MPI program to compute π MPI _Bcast and MPI _Reduce. Compare execution time for Serial code and Parallel code.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

double fX(double x)
{
  return (4 / (1 + (x * x)));
}

double fY(double y1, double y2, double h)
{
  return (0.5 * h * (y1 + y2));
}

int main(int argc, char *argv[])
{
  int myrank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  int N=atoi(argv[1]);  // tanking input from command line
  double lsum = 0;
  double I;
  MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Barrier(MPI_COMM_WORLD);
  int scattersize = (N / size);
  double X[N], x[scattersize], y[scattersize];
  double h = (1.00 / N);

  if (myrank == 0)
  {
    for (int i = 0; i < N; i++)
    {
      X[i] = (i * h);
    }
  }
  MPI_Scatter(&X, scattersize, MPI_DOUBLE, &x, scattersize, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
  double t1=MPI_Wtime();
  for (int i = 0; i < scattersize; i++)
  {
    y[i] = fX(x[i]);
  }

  for (int i = 1; i < scattersize; i++)
  {
    lsum += fY(y[i - 1], y[i], h);
  }
  double t2 = MPI_Wtime();
  printf("\nProcess: %d --> Time Required to find it's part of Pi : %1.10f \n", myrank,
(t2 - t1));
  MPI_Reduce(&lsum, &I, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
  double t3 = MPI_Wtime();
  MPI_Barrier(MPI_COMM_WORLD);
  if (myrank == 0 && size>1)
  {
    printf("\nProcess: %d --> Time Required to Perform final Reduce : %1.10f \n",
myrank, (t3 - t2));
    printf("Value of 'pi' for %d intervals is : \n%1.25f\n", N, I);
  }

  MPI_Finalize();

  return 0;
}
```

Output:

I) with n=1 (Serial) and steps = 50000

```
Process: 0 --> Time Required to find it's part of Pi : 0.0009060140

Value of 'pi' for 50000 intervals is :
3.1415526531231274809385923
```

II) with n=2 (Parallel) and steps = 50000

```
Process: 0 --> Time Required to find it's part of Pi : 0.0004047490

Process: 0 --> Time Required to Perform final Reduce : 0.0000293980
Value of 'pi' for 50000 intervals is :
```

```
Process: 1 --> Time Required to find it's part of Pi : 0.0004101780
3.141488652611134301686434
```

III) with n=4 (Parallel) and steps = 50000

```
Process: 1 --> Time Required to find it's part of Pi : 0.0003446080

Process: 0 --> Time Required to find it's part of Pi : 0.0004116410

Process: 0 --> Time Required to Perform final Reduce : 0.0000337610
Value of 'pi' for 50000 intervals is :
3.1413621576476415953038668

Process: 2 --> Time Required to find it's part of Pi : 0.0003332160

Process: 3 --> Time Required to find it's part of Pi : 0.0003373560
```

Conclusion:

From outputs it can be seen that, as number of processes increases the execution time goes on decreasing and also error in value goes on increasing.

Q8. Write a MPI program to compute Dot Product.

a) Each process gets an equal sized chunk of both the arrays (using MPI _Scatter).

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
int dotProduct(int a[],int b[],int size)
{
    int result=0;
    for(int i=0;i<size;i++)
    {
        result+=a[i]*b[i];
    }
    return result;
}
void printArray(int arr[], int r, int myrank)
{
    printf("Vector at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argc,char* argv[])
{
    int size,myrank,gresult;
    int N=atoi(argv[1]);  // tanking input from command line
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    int scattersize,newsize;
    if((N%size)==0)
    {
        scattersize=(N/size);
    }
    else
    {
        scattersize=(N/size)+1;
    }
    newsize=scattersize*size;
    int a[newsize];
    int b[newsize];
    if(myrank==0)
    {
        for(int i=0;i<newsize;i++)
        {
            if(i<N)
            {
            a[i]=rand()%(N*2);
            b[i]=rand()%(N*2);
            }
            else
            {
                a[i]=0;
                b[i]=0;
            }
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    int c[scattersize];
    int d[scattersize];

MPI_Scatter(&a,scattersize,MPI_INT,&c,scattersize,MPI_INT,0,MPI_COMM_WOR
LD);

MPI_Scatter(&b,scattersize,MPI_INT,&d,scattersize,MPI_INT,0,MPI_COMM_WOR
LD);
    double t1 = MPI_Wtime();
    int result=dotProduct(c,d,scattersize);
    double t2 = MPI_Wtime();
    printArray(c,scattersize,myrank);
    printArray(d,scattersize,myrank);
    printf("At process %d, dot product is %d.\n",myrank,result);
    printf("\nProcess: %d --> Time Required to find it's part of dot product : %1.10f \n",
myrank, (t2 - t1));
    MPI_Reduce(&result,&gresult,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
    double t3 = MPI_Wtime();
    if(myrank==0)
    {
        // printf("\nFirst vector is ");
        // printArray(a,N);
        // printf("\n");
        // printf("\nSecond vector is ");
        // printArray(b,N);
        // printf("\n");
        printf("\nFinal dot product is %d.\n\n",gresult);
        printf("\nProcess: %d --> Time Required to Perform final reduce: %1.10f \n",
myrank, (t3 - t2));
    }
    MPI_Finalize();
    return 0;
}
```

Output:

```
Vector at process 0 is 3  17  13  6  9
Vector at process 0 is 6  15  15  12  1
At process 0, dot product is 549.

Process: 0 --> Time Required to find it's part of dot product : 0.0000001720

Vector at process 1 is 2  10  3  0  12
Vector at process 1 is 7  19  6  6  16
At process 1, dot product is 414.

Process: 1 --> Time Required to find it's part of dot product : 0.0000002450
Final dot product is 963.


Process: 0 --> Time Required to Perform final reduce: 0.0000503530
```

b) Each process gets an unequal sized chunk of both the arrays (using MPI_Scatterv).

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
#define N 10
int dotProduct(int a[],int b[],int size)
{
    int result=0;
    for(int i=0;i<size;i++)
    {
        result+=a[i]*b[i];
    }
    return result;
}
void printArray(int arr[], int r,int myrank)
{
    printf("Vector at process %d is ",myrank);
    for (int i = 0; i <r; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
int main(int argv,char* argc[])
{
    int size,myrank,gresult;
    int a[N],b[N];
    int sum=0,result=0;
    MPI_Init(&argv,&argc);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    int sendcounts[]={4,6}; // Number of elements to be send to each processor
    int displs[]={0,4};     // relative to sendbuf from which to take the outgoing data to
process i
    if (myrank == 0)
    {
        for (int i = 0; i < N; i++)
        {
            a[i] = rand() % (N * 2);
            b[i] = rand() % (N * 2);
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    int c[sendcounts[myrank]],d[sendcounts[myrank]];

MPI_Scatterv(&a,sendcounts,displs,MPI_INT,&c,sendcounts[myrank],MPI_INT,0,M
PI_COMM_WORLD);

MPI_Scatterv(&b,sendcounts,displs,MPI_INT,&d,sendcounts[myrank],MPI_INT,0,M
PI_COMM_WORLD);
    double t1 = MPI_Wtime();
    for(int i=0;i<sendcounts[myrank];i++)
    {
        result+=c[i]*d[i];
    }
    double t2 = MPI_Wtime();
    printArray(c,sendcounts[myrank],myrank);
    printArray(d,sendcounts[myrank],myrank);
    printf("At process %d, dot product is %d.\n",myrank,result);
    printf("\nProcess: %d --> Time Required to find it's part of dot product : %1.10f \n",
myrank, (t2 - t1));
    MPI_Reduce(&result,&gresult,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
    double t3 = MPI_Wtime();
    if(myrank==0)
    {
        // printf("\nFirst vector is ");
        // printArray(a,N);
        // printf("\n");
        // printf("\nSecond vector is ");
        // printArray(b,N);
        // printf("\n");
        printf("\nFinal dot product is %d.\n",gresult);
        printf("\nProcess: %d --> Time Required to find it's part of dot product : %1.10f \
n", myrank, (t3 - t2));
    }
    MPI_Finalize();
    return 0;
}
```

Output:

```
Vector at process 0 is 3  17  13  6
Vector at process 0 is 6  15  15  12
At process 0, dot product is 540.

Process: 0 --> Time Required to find it's part of dot product : 0.0000001400

Final dot product is 963.
Vector at process 1 is 9  2  10  3  0  12
Vector at process 1 is 1  7  19  6  6  16
At process 1, dot product is 423.

Process: 1 --> Time Required to find it's part of dot product : 0.0000001430

Process: 0 --> Time Required to find it's part of dot product : 0.0000508800
```

========================THE END========================
========================