Parallel and Distributed Computing-CSE4001L
S. DHANYA ABHIRAMI
16BCE0965

Lab Slots: L9+L10 **Date:** 16th October 2018

ASSESSMENT 5

1. Write a MPI Program to perform binary search.

CODE

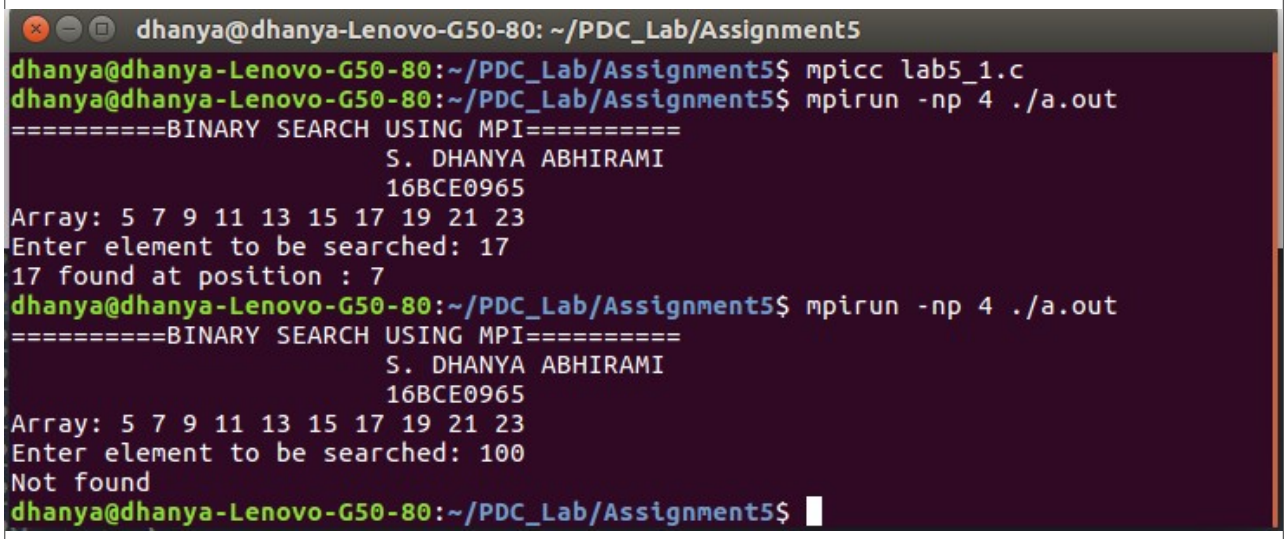
```
/*
==========MPI PROGRAM FOR BINARY SEARCH==========");
S. DHANYA ABHIRAMI
16BCE0965
Searching for an element in a array
*/
#include <stdio.h>
#include "mpi.h"
#include<stdlib.h>
int main(int argc,char *argv[])
{
    int rank, size;
    int n=10,i, first, middle, last, find;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    int* array = (int*)malloc(n*sizeof(int));
    for(i=0;i<n;i++)
    array[i]=i*2+5;
    if(rank==0)
    {
     printf("==========BINARY SEARCH USING MPI==========\n");
        printf("\t\t\tS. DHANYA ABHIRAMI\n\t\t\t16BCE0965\n");
        printf("Array: ");
     for(i=0;i<n;i++)
      printf("%d ",array[i] );
     printf("\nEnter element to be searched: ");
     scanf("%d", &find);
    }
    first = 0;
    last = n - 1;
    middle = (first+last)/2;
```

```
    while (first <= last) {
      if (array[middle] < find)
         first = middle + 1;
      else if (array[middle] == find) {
         printf("%d found at position : %d\n", find, middle+1);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
    }
    MPI_Finalize();
    if(rank==0){
      if (first > last)
      printf("Not found\n");
    }


    return 0;
}
```

OUTPUT



2. Write a MPI program to perform ring communication.

CODE

```
/*
==========MPI PROGRAM RING COMMUNICATION==========");
S. DHANYA ABHIRAMI
16BCE0965
Ring Communication
*/
#include  <stdio.h>
#include  <mpi.h>

void main(int argc, char *argv[])
```

```
{
  int rank, size, leftid, rightid;
  int val, sum, tmp;
  MPI_Status wait_status;
  MPI_Request recv_request;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  if (rank==0){
        printf("==========MPI PROGRAM RING
COMMUNICATION==========\n");
    printf("\t\t\t\t\tS. DHANYA
ABHIRAMI\n\t\t\t\t\t16BCE0965\n\n");
    }

  if ((leftid=(rank-1)) < 0) leftid = size-1;
  if ((rightid=(rank+1)) == size) rightid = 0;

  val = rank;
  sum = 0;
  do {
    MPI_Irecv(&tmp,1, MPI_INT, leftid, 99, MPI_COMM_WORLD,
&recv_request);

    MPI_Ssend(&val,1,MPI_INT,rightid,99, MPI_COMM_WORLD);

    MPI_Wait(&recv_request,&wait_status);

    val = tmp;
    sum += val;
  } while (val != rank);
  printf("\nProcess %d \nReceived from %d \nSends To %d Value
%d\n",rank,leftid,rightid,val);
  MPI_Finalize();

}
```

OUTPUT

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC_Lab/Assignment5
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpicc lab5_2b.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpirun -np 4 ./a.out
==========MPI PROGRAM RING COMMUNICATION==========
                                        S. DHANYA ABHIRAMI
                                        16BCE0965


Process 0
Received from 3
Sends To 1 Value 0

Process 1
Received from 0
Sends To 2 Value 1

Process 2
Received from 1
Sends To 3 Value 2

Process 3
Received from 2
Sends To 0 Value 3
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$
```

3. Write a MPI program to perform the squaring of numbers in array.

Input sequence: 2 4 8 16

Output sequence: 4 16 64 256

CODE

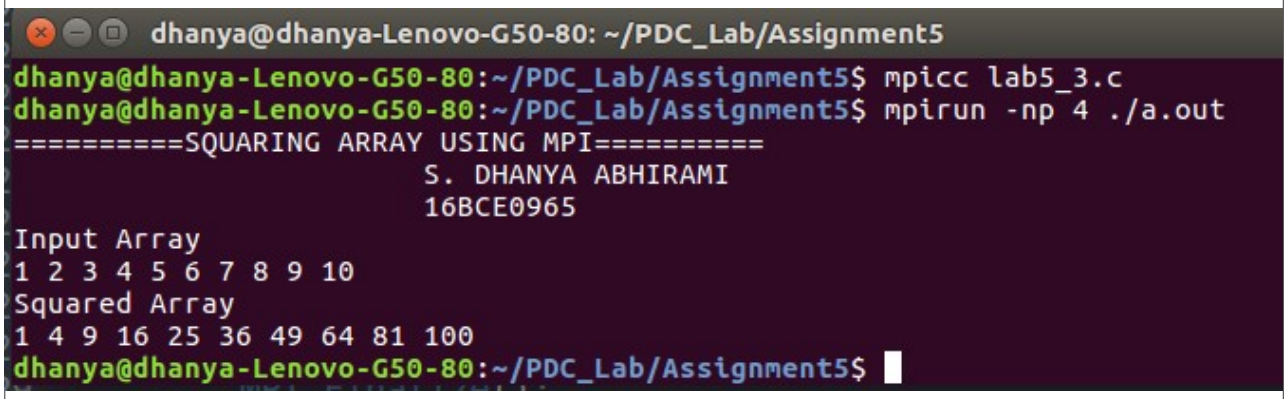
```
/*
==========MPI PROGRAM FOR SQUARING AN ARRAY==========");
S. DHANYA ABHIRAMI
16BCE0965
Squaring the elements in a array
*/
#include <stdio.h>
#include "mpi.h"
#include <stdlib.h>
int main(int argc,char *argv[])

    {
        int size,rank;
        int i,N=10;
        int* array = (int*)malloc(N*sizeof(int));
        int* sq_array = (int*)malloc(N*sizeof(int));
        if(rank==0)
        {
            for (i = 0; i < N; i++)
            array[i] = i+1;
        }
        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        for(i=0;i<N;i++)
```

```
        {
            sq_array[i]=array[i]*array[i];
        }
        MPI_Finalize();
        if(rank==0)
        {
            printf("==========SQUARING ARRAY USING
MPI==========\n");
        printf("\t\t\tS. DHANYA ABHIRAMI\n\t\t\t16BCE0965\n");
        printf("Input Array\n");
            for(i=0;i<N;i++)
        printf("%d ",array[i]);
        printf("\nSquared Array\n");
            for(i=0;i<N;i++)
        printf("%d ",sq_array[i]);
        printf("\n");}
        return 0;

    }
```

OUTPUT



4. Write a MPI program to perform the sum of 1000 numbers using gather and scatter.

CODE

```
/*
==========MPI SUM USING GATHER SCATTER PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
Calculating sum of first 1000 natural numbers
*/
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char *argv[]){
    int rank, size, i ;
    int n ,partial_sum, N=1000, global_sum=0 ;
    // Initialisig MPI Environment
    MPI_Init( &argc , &argv );

    // Getting number of processors
```

```
    MPI_Comm_size(MPI_COMM_WORLD , &size);

    // Getting Rank of process
    MPI_Comm_rank(MPI_COMM_WORLD , &rank);

    // The array
    int* array = (int*)malloc(N*sizeof(int));
    if(rank==0){
    for (i = 0; i < N; i++) {
        array[i] = i+1;
    }
    printf("==========SUM OF 1000 NATURAL NUMBERS==========\n");
    printf("\t\t\tS. DHANYA ABHIRAMI\n\t\t\t16BCE0965\n\n");
    }


    n = N/size;
    int* sub_array = (int*)malloc(n*sizeof(int));
    int* sums = (int*)malloc(size*sizeof(int));

    // Scatter sub arrays to processess

MPI_Scatter(array,n,MPI_INT,sub_array,n,MPI_INT,0,MPI_COMM_WORLD);

    // Calculate partial sum of the sub array
    partial_sum = 0;
    for(i=0;i<n;i++)
        partial_sum+=sub_array[i];
    printf("Process %d Partial Sum = %d\n",rank,partial_sum);
    // Combine the partial sums into an array
    MPI_Gather(&partial_sum,1,MPI_INT, sums,
1,MPI_INT,0,MPI_COMM_WORLD);

    if(rank == 0)
    {
        for(i=0;i<size;i++)
            {
                global_sum+=sums[i];
            }
        printf("\n\nGlobal Sum = %d\n",global_sum );
    }

    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
}
```

OUTPUT

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC_Lab/Assignment5
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpicc lab5_4.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpirun -np 4 ./a.out
==========SUM OF 1000 NATURAL NUMBERS==========
                        S. DHANYA ABHIRAMI
                        16BCE0965

Process 0 Partial Sum = 31375
Process 1 Partial Sum = 93875
Process 2 Partial Sum = 156375
Process 3 Partial Sum = 218875


Global Sum = 500500
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$
```

5. Write a MPI program to perform the sum of 1000 numbers using MPI broadcast and reduce function. Calculate the time using MPI wall time function.

CODE

```
/*
==========MPI SUM USING BROADCAST AND REDUCE PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
Calculating sum of first 1000 natural numbers
*/
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#include<stdlib.h>
int main(int argc, char **argv)
{
     int rank, size,num, N=1000;
     int i, n, low, high, partial_sum=0, global_sum;
     double startwtime,endwtime;
     int* array = (int*)malloc(N*sizeof(int));
     MPI_Init(&argc, &argv);
     MPI_Comm_size(MPI_COMM_WORLD, &size);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);

     if(rank==0) {
          for(i=0; i<N; i++) {
               array[i]=i+1;
          }
          printf("==========SUM OF 1000 NATURAL
NUMBERS==========\n");
        printf("\t\t\tS. DHANYA ABHIRAMI\n\t\t\t16BCE0965\n");
          startwtime = MPI_Wtime();
     }

     MPI_Bcast(array, N, MPI_INT, 0, MPI_COMM_WORLD);

     n = N/size;
```

```
     low = rank * n;
     high = low + n;
     for(i=low; i<high; i++) {
          partial_sum += array[i];
     }
     printf("Process %d Partial Sum %d\n", rank,partial_sum);

     /* compute global sum */
     MPI_Reduce(&partial_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

     if(0 == rank) {
          endwtime = MPI_Wtime();
          printf("Global Sum = %d\n", global_sum);
          printf("Wall Clock Time: %lf\n",endwtime- startwtime );
     }

     MPI_Finalize();
}
```

OUTPUT

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC_Lab/Assignment5
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpicc lab5_5.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpirun -np 4 ./a.out
==========SUM OF 1000 NATURAL NUMBERS==========
                        S. DHANYA ABHIRAMI
                        16BCE0965
Process 0 Partial Sum 31375
Process 2 Partial Sum 156375
Process 1 Partial Sum 93875
Process 3 Partial Sum 218875
Global Sum = 500500
Wall Clock Time: 0.000203
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$
```

6. Write a MPI program to calculate the the value of pi using broadcast and reduce functions.

CODE

```
/*
==========MPI PROGRAM TO CALCULATE VALUE OF PI==========");
S. DHANYA ABHIRAMI
16BCE0965
tan(pi/4) = 1
arctan(1) = pi/4
d(arctan(x))/dx = 1.0/(1+x*x)
Performing numerical integration and multiplying by 4
*/
#include <stdio.h>
#include <mpi.h>
#include <math.h>
```

```
int main(int argc, char *argv[])
{
    int n;
    double PI25DT = 3.141592653589793238462643;
    double local_value, h, pi, i, sum, x, dx_arctan, startwtime,
endwtime;
    int rank, size, resultlen;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if (rank==0){
        printf("==========APPROXIMATING VALUE OF PI USING
BROADCAST AND REDUCE==========\n");
    printf("\t\t\t\t\tS. DHANYA
ABHIRAMI\n\t\t\t\t\t16BCE0965\n\n");
    }
    if(rank == 0)
    {
    // n = number of evaluation points
    n = 100000;
    printf("The number of intervals = %d \n", n);
    startwtime = MPI_Wtime();
    }

    // Share intervals with other processors
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    sum = 0.0;
    h   = 1.0/n;
    // Computing and Adding the "Heights" of each bar of the
integration
    for (i=rank+0.5; i<n; i+=size)
    {
        // Derivative of arctan
        dx_arctan = 1.0 / (1.0 + (i*h)*(i*h));
        sum += dx_arctan;
    }
    // Multiplying by the "Widths" of each bar and 4.0
(arctan(1)=Pi/4)
    local_value = 4.0*h*sum;
    /* Show all processor IDs */
    printf("Process-%d Local value = %lf \n",rank,local_value);
    // Consolidate and Sum Results
    MPI_Reduce(&local_value, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

    if (rank == 0)
    {
        endwtime = MPI_Wtime();
        printf("pi = (approx) %.16f\nError is %.16f\n",pi, fabs(pi
- PI25DT));
```

```
        printf("Execution Wall clock time = %f\n", endwtime-
startwtime);
    }
    MPI_Finalize();
    return 0;
}
```

OUTPUT

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC_Lab/Assignment5
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpicc lab5_6.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$ mpirun -np 4 ./a.out
==========APPROXIMATING VALUE OF PI USING BROADCAST AND REDUCE==========
                                        S. DHANYA ABHIRAMI
                                        16BCE0965

The number of intervals = 100000
Process-1 Local value = 0.785401
Process-2 Local value = 0.785396
Process-0 Local value = 0.785406
Process-3 Local value = 0.785391
pi = (approx) 3.1415926535981167
Error is 0.0000000000083236
Execution Wall clock time = 0.000392
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment5$
```