VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

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

Lab Slots: L9+L10 **Date:** 9th October 2018

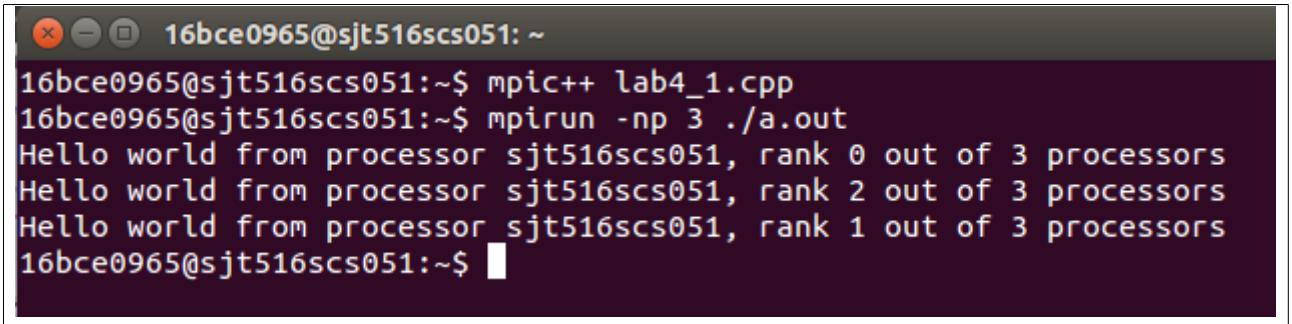
ASSESSMENT 4

1. Write a sample hello world program using MPI functions. Describe the MPI functions with the syntax.

Code

```
/*
==========MPI HELLO WORLD PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
*/
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    // Initialisig MPI Environment
    MPI_Init( &argc, &argv );
    // Getting Rank of process
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    // Getting number of processors
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    // Getting Name of Processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name,&name_len);
    printf( "Hello world from processor %s, rank %d out of %d
processors\n", processor_name, rank, size );
    // Finalising MPI environment
    MPI_Finalize();
    return 0;
}
```

Output



Function Explanation

a. int MPI_Init(int *argc, char ***argv)

MPI_Init is used to Initialize the MPI execution environment

Input Parameters

argc - Pointer to the number of arguments

argv -Pointer to the argument vector

b. int MPI_Comm_rank(MPI_Comm comm, int *rank)

MPI_Comm_rank determines the rank of the calling process in the communicator

Input Parameters

comm - communicator (handle)

Output Parameters

rank - rank of the calling process in the group of `comm` (integer)

c. int MPI_Comm_size(MPI_Comm comm, int *size)

MPI_Comm_size determines the size of the group associated with a communicator

Input Parameters

comm - communicator (handle)

Output Parameters

size - number of processes in the group of `comm` (integer)

d. int MPI_Get_processor_name(char *name, int *resultlen)

MPI_Get_processor_name Gets the name of the processor

Output Parameters

name - A unique specifier for the actual (as opposed to virtual) node. This must be an array of size at least `MPI_MAX_PROCESSOR_NAME`.

resultlen - Length (in characters) of the name

e. int MPI_Finalize(void)

MPI_Finalize Terminates MPI execution environment .

All processes must call this routine before exiting.

f. MPI_COMM_WORLD

Integer representing a pre-defined communicator consisting of all processes.

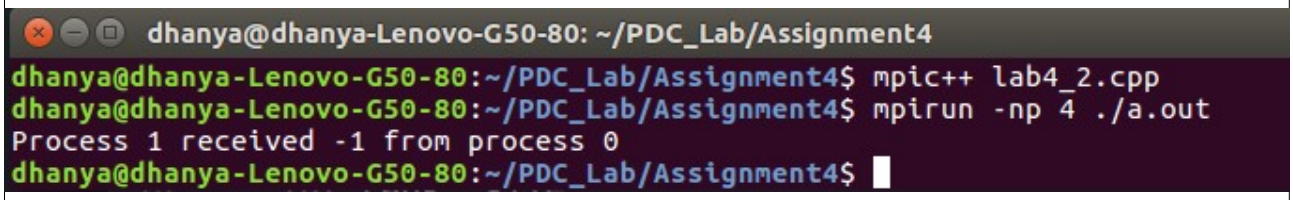
2. Write a MPI program to show the usage of send and receive commands used in MPI program. Describe the functions used.

Code

```
/*
==========MPI SEND AND RECEIVE PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
*/
#include<mpi.h>
#include<stdlib.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
    // Initialisig MPI Environment
    MPI_Init( &argc, &argv );
    int rank, size;
    // Getting Rank of process
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    // Getting number of processors
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    // Assuming at least 2 processess for this task
    if(size<2){
        fprintf(stderr,"World size must be greater than 1 for
%s\n",argv[0]);
        MPI_Abort(MPI_COMM_WORLD,1);
    }
    int number;
    // If rank = 0, set number to -1 and send it to process 1
    if(rank==0){
        number=-1;
        MPI_Send(&number,1,MPI_INT,1,0,MPI_COMM_WORLD);
    }
    else if(rank==1){

MPI_Recv(&number,1,MPI_INT,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```

```
     printf("Process 1 received %d from process 0\n",number);
    }
    MPI_Finalize();
    return 0;
}
```

Output



Function Explanation

a. int MPI_Abort(MPI_Comm comm, int errorcode)

MPI_Abort terminates MPI execution environment

Input Parameters

comm - communicator of tasks to abort

errorcode - error code to return to invoking environment

b. int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

MPI_Send performs a blocking send

Input Parameters

buf - initial address of send buffer (choice)

count - number of elements in send buffer (nonnegative integer)

datatype - datatype of each send buffer element (handle)

dest - rank of destination (integer)

tag - message tag (integer)

comm - communicator (handle)

c. int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm, MPI_Status *status)

MPI_Recv performs blocking receive for a message

Output Parameters

buf - initial address of receive buffer (choice)

status - status object (Status)

Input Parameters

count - maximum number of elements in receive buffer (integer)

datatype - datatype of each receive buffer element (handle)

source - rank of source (integer)

tag - message tag (integer)

comm - communicator (handle)

3. Write a MPI program to find the dot product of the vector. Use MPI reduce function to combine all the result and describe the functionality of reduce function.

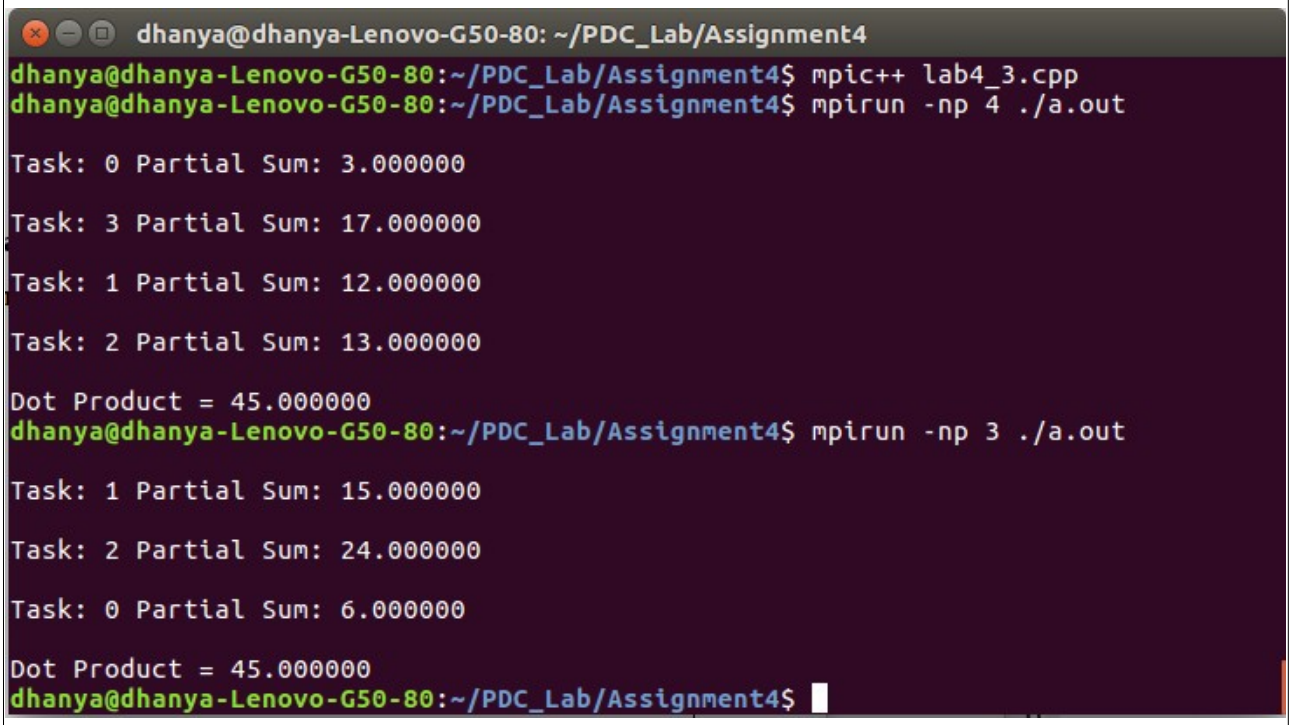
Code

```
/*
==========MPI DOT PRODUCT PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
*/
#include<mpi.h>
#include<stdlib.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
    // Initialisig MPI Environment
    MPI_Init( &argc, &argv );
    int rank, size;

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

    // Getting number of processors
    MPI_Comm_size( MPI_COMM_WORLD, &size );

    if(size<2){
```

```
    fprintf(stderr,"World size must be greater than 1 for
%s\n",argv[0]);
    MPI_Abort(MPI_COMM_WORLD,1);
    }
    int i,len=10;
    double dotprod=0.0,partial_dotprod=0.0;
    double *vec1,*vec2;
    vec1 = (double*)malloc(len*sizeof(double));
    vec2 = (double*)malloc(len*sizeof(double));

    // Rescaling the parameters
    int startval = len * rank/size + 1;
    int endval = len * (rank+1) / size;

    // Assigning values to vectors
    for(i=startval;i<=endval;i++){
    vec1[i]=1.0; // vec1 =
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
    vec2[i]=i*1.;// vec2=[0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
    }

    // Computing Partial Dot Product
    for(i=0;i<len;i++){
    partial_dotprod+=vec1[i]*vec2[i];
    }
    printf("\nTask: %d Partial Sum: %f\n",rank,partial_dotprod);

    // Combining Partial Products to Global Dot Product

MPI_Reduce(&partial_dotprod,&dotprod,1,MPI_DOUBLE,MPI_SUM,0,MPI_CO
MM_WORLD);

    if(rank==0){
    printf("\nDot Product = %f\n",dotprod);}

    // Blocks until all processes in the communicator have reached
this routine
```

```
    MPI_Barrier(MPI_COMM_WORLD);

    // Finalising MPI environment
    MPI_Finalize();
    return 0;
}
```

Output



Function Explanation

a. int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

MPI_Reduce reduces values on all processes to a single value

Input Parameters

sendbuf - address of send buffer (choice)

count - number of elements in send buffer (integer)

datatype - data type of elements of send buffer (handle)

op - reduce operation (handle)

root - rank of root process (integer)

comm - communicator (handle)

Output Parameters

recvbuf - address of receive buffer (choice, significant only at `root`)

b. int MPI_Barrier(MPI_Comm comm)

MPI_Barrier blocks until all processes in the communicator have reached this routine. It blocks the caller until all processes in the communicator have called it; that is, the call returns at any process only after all members of the communicator have entered the call.

Input Parameters

Comm - communicator (handle)

c. MPI_DOUBLE

Represents Datatype double

d. MPI_SUM

Computes Sum of all values

4. Write a MPI program to find the average of an array of elements. Use MPI reduce function and describe the function.

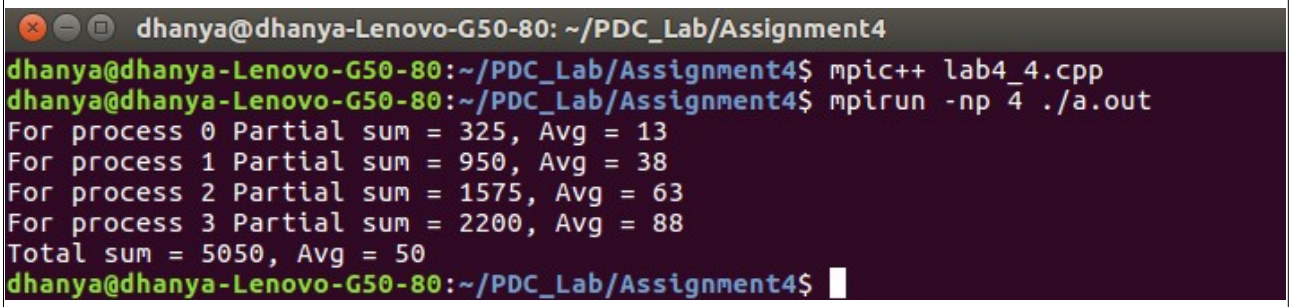
Code

```
/*
==========MPI AVERAGE PROGRAM==========");
S. DHANYA ABHIRAMI
16BCE0965
Calculating average of first 100 natural numbers
*/
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char *argv[]){
    int rank, size, i ;
    int n , begin, end, N=100 ;

    // 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);

    // Rescaling the parameters
    begin = N * rank/size + 1;
    end = N * (rank+1) / size;

    n = N / size;

    // The array
    int* array = (int*)malloc(N*sizeof(int));
    for (i = begin; i <= end; i++) {
        array[i] = i;
    }

    // Find the partial sum
    int partial_sum = 0;

    for (i = begin; i <= end; i++) {
        partial_sum = partial_sum+array[i];
    }

    // Print the random numbers on each process
    printf("For process %d Partial sum = %d, Avg = %d\n",
        rank, partial_sum, partial_sum / n);



    // Using MPI_Reduce to reduce the ocal sums into global sum
    int global_sum=0;
    MPI_Reduce(&partial_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
               MPI_COMM_WORLD);

    // Computing Average
    if (rank == 0) {
       printf("Total sum = %d, Avg = %d\n", global_sum,
```

```
        global_sum/(size*n));
    }

    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
}
```

Output



Function Explanation

a. int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

MPI_Reduce reduces values on all processes to a single value

Input Parameters

sendbuf - address of send buffer (choice)

count - number of elements in send buffer (integer)

datatype - data type of elements of send buffer (handle)

op - reduce operation (handle)

root - rank of root process (integer)

comm - communicator (handle)

Output Parameters

recvbuf - address of receive buffer (choice, significant only at `root`)

b. int MPI_Barrier(MPI_Comm comm)

MPI_Barrier blocks until all processes in the communicator have reached this routine. It blocks the caller until all processes in the communicator have called it; that is, the call returns at any process only after all members of the communicator have entered the call.

Input Parameters

Comm - communicator (handle)

c. MPI_INT

Represents Datatype integer

d. MPI_SUM

Computes Sum of all values