**Name: Abinash Satapathy**

**Reg. No.: 16BCE0081**

**Slot: L45 + L46**

**Subject: Parallel & Distributed Computing (CSE4001) Lab**

**Experiment – 3**

**Functions used**:

* **MPI\_Init**

Initialize the MPI execution environment

* **MPI\_Finalize**

Terminates MPI execution environment

* **MPI\_Comm\_size**

Determines the size of the group associated with a communicator

* **MPI\_Comm\_rank**

Determines the rank of the calling process in the communicator

* **MPI\_Send**

Performs a blocking send

* **MPI\_Recv**

Blocking receive for a message

* **MPI\_Reduce**

Reduces values on all processes to a single value

* **MPI\_Barrier**

Blocks until all processes in the communicator have reached this routine.

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

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// Get the rank of the process

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

// Get the name of the processor

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

// Print off a hello world message

printf("Hello world from processor %s, rank %d out of %d processors\n",

processor\_name, world\_rank, world\_size);

// Finalize the MPI environment.

MPI\_Finalize();

}

A screenshot of a computer

Description generated with very high confidence

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

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv){

MPI\_Init(NULL, NULL);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

int number;

if (world\_rank == 0) {

number = -1;

MPI\_Send(&number, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

} else if (world\_rank == 1) {

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

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

MPI\_Finalize();

}

}

A screenshot of a computer

Description generated with very high confidence

1. Write an 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.

#include <mpi.h>

#include <stdio.h>

const int N=2000;

double dotProduct(double \*x, double \*y, int n) {

int i;

double prod = 0.0;

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

prod += x[i]\*y[i];

}

return prod;

}

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

int i;

double prod;

int my\_rank;

int num\_procs;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

int local\_N = N / num\_procs; //assuming N is totally divisible by num\_procs

double local\_x[local\_N];

double local\_y[local\_N];

for(i = 0; i < local\_N; i++) {

local\_x[i] = 0.01 \* (i + my\_rank \* local\_N);

local\_y[i] = 0.03 \* (i + my\_rank \* local\_N);

}

double local\_prod;

local\_prod = dotProduct(local\_x,local\_y,local\_N);

MPI\_Reduce(&local\_prod, &prod, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (my\_rank == 0) {

printf("dotProduct = %f\n", prod);

}

MPI\_Finalize();

return 0;

}

A screenshot of a computer

Description generated with very high confidence

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

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <time.h>

float \*create\_rand\_nums(int num\_elements){

float \*rand\_nums = (float \*) malloc(sizeof(float) \* num\_elements);

assert(rand\_nums != NULL);

int i;

for(i=0;i<num\_elements;i++)

rand\_nums[i] = (rand()/(float)RAND\_MAX);

return rand\_nums;

}

int main(){

int num\_elements\_per\_proc = 4;

MPI\_Init(NULL, NULL);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

float \*rand\_nums = NULL;

rand\_nums = create\_rand\_nums(num\_elements\_per\_proc);

// Sum the numbers locally

float local\_sum = 0;

int i;

for (i = 0; i < num\_elements\_per\_proc; i++) {

local\_sum += rand\_nums[i];

}

// Print the random numbers on each process

printf("Local sum for process %d - %f, avg = %f\n",

world\_rank, local\_sum, local\_sum / num\_elements\_per\_proc);

// Reduce all of the local sums into the global sum

float global\_sum;

MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_FLOAT, MPI\_SUM, 0,

MPI\_COMM\_WORLD);

// Print the result

if (world\_rank == 0) {

printf("Total sum = %f, avg = %f\n", global\_sum,

global\_sum / (world\_size \* num\_elements\_per\_proc));

}

free(rand\_nums);

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

}

A screenshot of a computer

Description generated with very high confidence