Name: Abinash Satapathy

Reg. No.: 16BCE0081

Slot: L45 + L46

Subject: Parallel & Distributed Computing (CSE4001) Lab

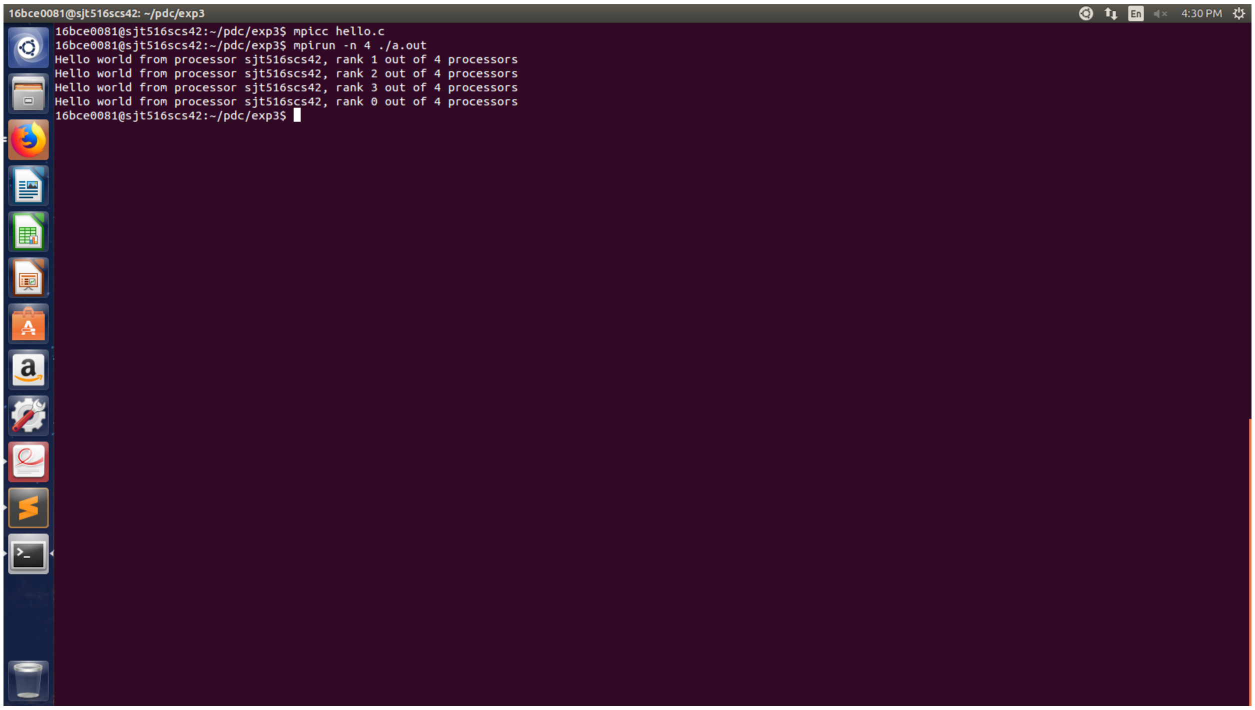
Experiment – 3

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.

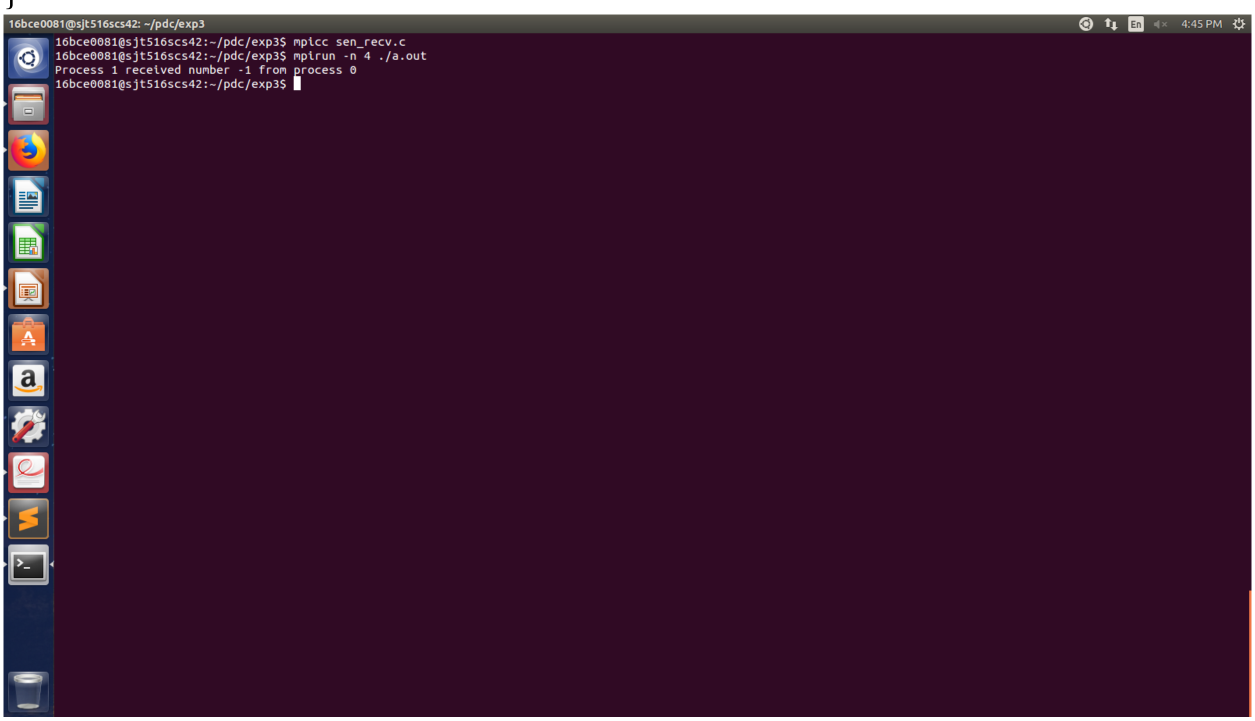
1. Write a sample hello world program using MPI functions. Describe the MPI functions with the syntax.

```
#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();
}
```


16bce0081@sjt516scs42: ~/pdc/exp3
4:30 PM
16bce0081@sjt516scs42:~/pdc/exp3$ mpicc hello.c
16bce0081@sjt516scs42:~/pdc/exp3$ mpirun -n 4 ./a.out
Hello world from processor sjt516scs42, rank 1 out of 4 processors
Hello world from processor sjt516scs42, rank 2 out of 4 processors
Hello world from processor sjt516scs42, rank 3 out of 4 processors
Hello world from processor sjt516scs42, rank 0 out of 4 processors
16bce0081@sjt516scs42:~/pdc/exp3$

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

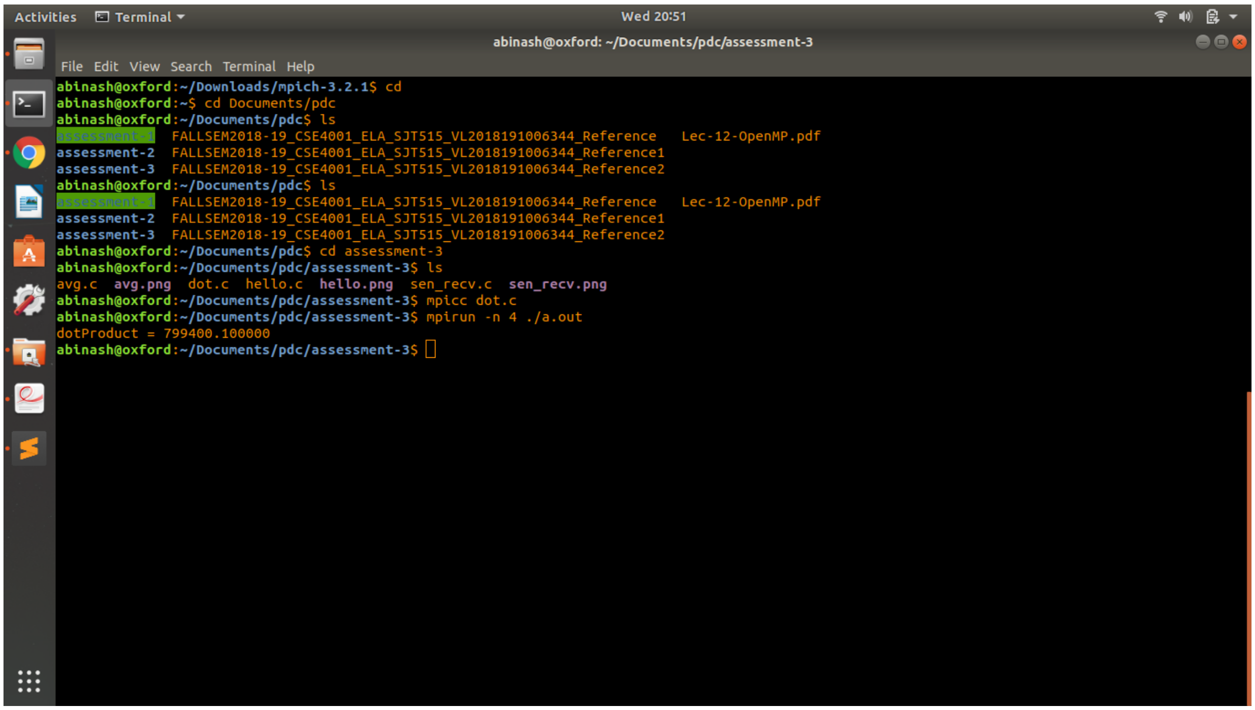
```
#include <mpi.h>
#include <stdio.h>

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();
}
}
```



3. 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;
}
```


Activities
Terminal
Wed 20:51
abinash@oxford: ~/Documents/pdc/assessment-3
File Edit View Search Terminal Help
abinash@oxford:~/Downloads/mpich-3.2.1$ cd
abinash@oxford:~$ cd Documents/pdc
abinash@oxford:~/Documents/pdc$ ls
assessment-1 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference Lec-12-OpenMP.pdf
assessment-2 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference1
assessment-3 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference2
abinash@oxford:~/Documents/pdc$ ls
assessment-1 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference Lec-12-OpenMP.pdf
assessment-2 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference1
assessment-3 FALLSEM2018-19_CSE4001_ELA_SJT515_VL2018191006344_Reference2
abinash@oxford:~/Documents/pdc$ cd assessment-3
abinash@oxford:~/Documents/pdc/assessment-3$ ls
avg.c avg.png dot.c hello.c hello.png sen_recv.c sen_recv.png
abinash@oxford:~/Documents/pdc/assessment-3$ mpicc dot.c
abinash@oxford:~/Documents/pdc/assessment-3$ mpirun -n 4 ./a.out
dotProduct = 799400.100000
abinash@oxford:~/Documents/pdc/assessment-3$

4. 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();

}
```