ASSIGNMENT HPC-4

Roll No: 41205

Problem Statement:

Design and implement parallel algorithm utilizing all resources available. for

• Binary Search for Sorted Array

• Best-First Search that (traversal of graph to reach a target in the shortest possible path)

Objective:

1. To under basics of MPI
2. Apply parallel programming concepts and write binary search and best first search algorithms

Outcome: One will be able to write parallel programs to perform complex algorithms using MPI

Pre-requisites:

1. 64-bit Linux OS
2. Programming Languages: C/C++

Hardware Specification:

1. x86_64 bit
2. 2/4 GB DDR RAM
3. 80 - 500 GB SATA HD
4. 1GB NIDIA TITAN X Graphics Card

Software Specification:

1. Ubuntu 14.04

Theory:

- Parallel programs enable users to fully utilize the multi-node structure of supercomputing clusters.
- Message Passing Interface (MPI) is a standard used to allow several different processors on a cluster to communicate with each other.
- To get started with MPI, there are four important functions:
- MPI_Init():
    - This function initializes the MPI environment. It takes in the addresses of the C++ command line arguments argc and argv.
- MPI_Comm_size():

    - This function returns the total size of the environment via quantity of processes. The function takes in the MPI environment, and the memory address of an integer variable.
- MPI_Comm_rank():
    - This function returns the process id of the processor that called the function. The function takes in the MPI environment, and the memory address of an integer variable.
- MPI_Finalize():
    - This function cleans up the MPI environment and ends MPI communications.

Syntax:

```
Sample hello world program:
#include "mpi.h"

int main(int argc, char **argv)
{
   int rank, size;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   printf("Hello world! Process of rank %d out of %d
processes.\n",rank,size);
   MPI_Finalize();
   return 0;
}

Running the program:
mpic++ file.c
mpirun -np 5 ./a.out
```

Test Cases:

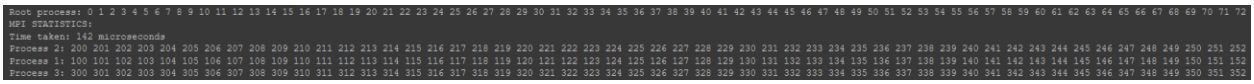
#	Input	Expected Output	Actual Output	Result
1	Perform binary search	Element found	Element found	Success
2	Perform best first search	Search performed and elements printed	Search performed and elements printed	Success

Output:

Binary Search

```
MPI STATISTICS:
Time taken: 1559 microseconds
Element found at process 9 at: 9912.
```

Best First Search



Conclusion: Thus, we were able to perform binary search and best first search using MPI in C++.