

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

```
Root process: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72  
MPI STATISTICS:  
Time taken: 142 microseconds  
Process 2: 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252  
Process 1: 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152  
Process 3: 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352
```

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