­­­­

GOVERNMENT COLLEGE UNIVERSITY, LAHORE­­­­­­­

DEPARTMENT OF COMPUTER SCIENCE

**Report: Quick Sort Parallel Algorithm**

|  |  |
| --- | --- |
| **SUBMITTED BY:** | Muhammad Zeeshan Yousaf |
| **ROLL NO:** | 259-BSCS-19 |
| **SECTION:** | E1 |
| **SEMESTER:** | 7th |
| **COURSE:** | Parallel and Distributed Computing |
| **SUBMITTED TO:** | Sir Umair Sadiq |

Parallel Performance of MPI Quick Sort Algorithm

# Abstract

Message Passing Interface (**MPI**) is widely used to implement parallel programs. Although Windows based architectures provide the facilities of parallel execution and multi-threading, little attention has been focused on using MPI on Linux platforms. In this report I use the core i7 Ubuntu-based platform to study the effect of parallel processes on the performance of MPI parallel implementations for Quick sort algorithm.

# Keywords

*Parallel programming, Message Passing Interface, performance*

# Introduction

QuickSort is a Divide and Conquer Algorithm. It picks an element as a pivot and partitions the array around the picked pivot. There are many ways of choosing the pivot elements. They are:

* Always pick the first element as a pivot.
* Always pick the last element as the pivot (implemented below)
* Pick a random element as a pivot.
* Pick median as a pivot.

MPI: MPI stands for Message Passing Interface. Here the message is data. MPI allows data to be passed between processes in a distributed memory environment. In C, “mpi.h” is a header file that includes all data structures, routines, and constants of MPI. Using “mpi.h” parallelized the quick sort algorithm. Below is the C program to implement quicksort using MPI:

|  |
| --- |
| */\* Quick Sort \*/* // (quick) sort slice of array v; slice starts at s and is of length n 1. quicksort(sub-list v, start, length) 2.{ 3. int pivot, pstart, i; 4. if (length <= 1) 5. return; // pick pivot and swap with first element 6. pivot = sub-list[start + length/2]; 7. swap(sub-list[], start, s + length/2); // partition slice starting at s+1 8. pstart = start; 9. for (i = start+1; i < start+length; i++) 10. if (sub-list[i] < pivot) 11. { pstart++; 12. swap(sub-list[], i, pstart); 13. } // swap pivot into place 14. swap(sub-list[], start, pstart); // recurse into partition 15. quicksort(sub-list[], start, pstart-start); 16. quicksort(sub-list[], pstart+1, start+length-pstart-1); 17.} |

### The MPI code for Quick Sort is [here](https://github.com/MuhammadZeeshanYousaf/Parallel-Programming-MPI/tree/main/Project) on my Github

<https://github.com/MuhammadZeeshanYousaf/Parallel-Programming-MPI/tree/main/Project>

# Result of experiments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No. of Processors (P)** | **Data (Integers to sort)** | **Execution time (sec)** | **Speed up ( Ts / Tp )** | **Efficiency ( S / P )** |
| 1 | 13000 | 4.203 | 1 | 1 |
| 2 | 13000 | 3.219 | 1.3056 | 0.6528 |
| 4 | 13000 | 3.328 | 1.2629 | 0.3157 |
| 8 | 13000 | 3.153 | 1.3330 | 0.1666 |
| 12 | 13000 | 3.056 | 1.3753 | 0.1146 |
| 16 | 13000 | 3.131 | 1.3423 | 0.0838 |

# Speed up Graph

# Efficiency Graph

# Conclusion

In this report I focused on using MPICH as a message passing interface implementation on  
Ubuntu platforms. The effect of parallel processes number and also the number of cores on  
the performance of parallel quick sort algorithms has been experimentally studied.  
I found that the computation / communication ratio greatly affects the execution time of the  
three studied parallel algorithms.