PARALLEL AND DISTRIBUTED COMPUTING CS-3006

**Analysis of sorting algorithms**

A picture containing text, gear, metalware

Description automatically generated

SUBMITTED TO:

Sir Nadeem Kafi

GROUP MEMBERS:

HUZAIFA TANZEEL **20K-0305**

YUSRA ADAM **20K-0207**

**INTRODUCTION:**

In today’s world of problems, sorting algorithms are important as they reduce complexities of various problems. Sorting algorithms are used in many real-world applications like searching techniques, database algorithms, data structure problem, divide and conquer concepts and many more. Any sorting algorithm can be implemented depending upon the problem requirement. There have been a lot of work done on which sorting algorithm gives most ideal results in terms of complexities.

We have also tested sorting algorithms to analyze its performance and complexity serially and through parallel approach.

**OBJECTIVE:**

Data parallelism is one of the main aspects of parallel computing. Data parallelism is a way of distributing data among different processors or nodes and then computation will be performed on each data set in parallel.

In this project, we will use various sorting algorithms to explain data parallelism, in which large data set will be distributed to multiple processors or nodes and then sorting will be performed by each individual processor in parallel. Furthermore, demonstration of various analysis will be done in terms of time complexities, performance and efficiency.

**SERIAL PROGRAMMING:**

Serial programming in computer architecture means that a single task will be executed at a time and all tasks are executed by the processor in sequence. The performance of computer architecture is determined by the processer it has which is responsible for executing instructions and performing the tasks assigned to it. Serial machines have only one processor and hence the performance is slower as execution is done sequentially. As there is only one processor therefore workload on the processor is high.

**PARALLEL PROGRAMMING:**

As compared to serial programming, parallel programming in computer architecture means that execution of several tasks will be done at one time. In parallel processing there are multiple processors and each processor performs its assigned tasks simultaneously. As there are multiple processors hence the performance is higher and work load on each processor is low and balanced.

**PARALLEL PROGRAMMING USING OPENMP AND MPI:**

Openmp in parallel programming is a library for execution of tasks in parallel manner. In Openmp there is shared memory which is parallel programming on a single node. With Openmp all threads share memory and data. There are directives for an Openmp program to work in parallel which the programmer needs to specify. What region of code should perform in parallel, variables assignment as shared or private, threads synchronization.

On the other hand, MPI in parallel programming is a message passing paradigm between multiple computers that runs a parallel program across distributed memory. We can execute a parallel program on a cluster of multiprocessors by executing MPI program in each processor.

**WHY PARALLEL PROGRAMMING:**

Now the question arises that why we need parallel computation? And the answer is obvious which to avoid huge time computation and low performance while working with serial programming. In case of serial programming, if processor gets failure, then whole execution will suffer. But that’s not the case in parallel as there are more than one processor and all processors work independently and failure in any one processor does not affect the functionality and working of other processors.

Through implementation of sorting algorithms, we have analyzed how much time varies when working with serial programming and parallel programming.

**IMPLEMENTATION:**

We have implemented our idea in a way that we selected few serial sorting algorithms and implemented them on a huge data set as analyses can be done better in case of huge data. The data set is of 100 million and 500 million of randomly generated numbers. We calculated the time it took to sort these numbers so that we can think of reducing the time by implementing them in parallel.

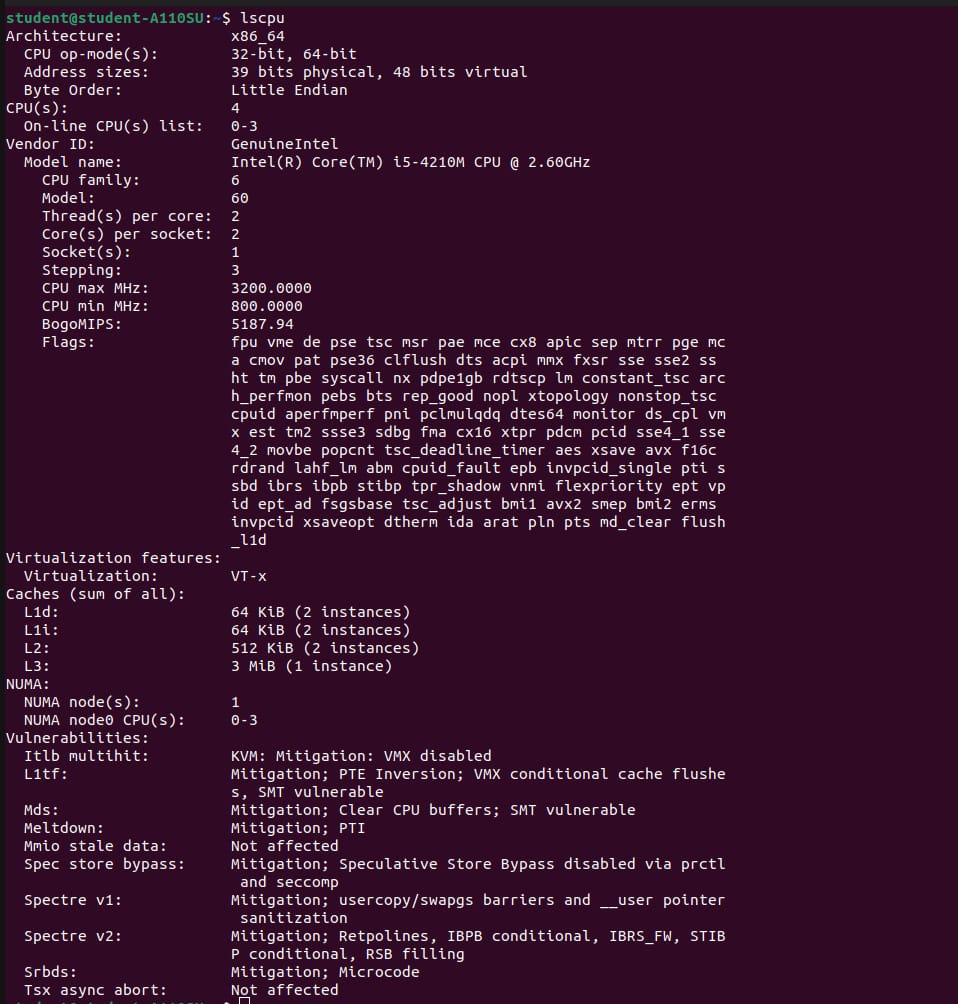
For parallel implementation, we executed the serial version of sorting algorithms in Openmp and MPI on different number of threads and processes respectively on the dataset of above-mentioned size. Then we performed analysis about the time taken by serial execution and parallel execution.

**WHAT PROBLEM IS IT SOLVING:**

Sorting algorithms can be used to maintain the accuracy of your problem and reduce algorithmic complexity of any problem. In case of big data set, implementation of any searching technique in unordered data is difficult, so by sorting the data optimum searching techniques can be used to target the required data. Apart from this, databases use sorting techniques to load large amount of data into memory.

In distributed systems we have to divide the tasks among the processors and merge the results back so divide n conquer algorithms are used in parallel using MPI.

**ARCHITECTURE DETAILS:**



**ANALYSIS OF MERGE SORT USING SERIAL AND PARALLEL PROGRAMMING:**

**WHAT IS MERGE SORT:**

Merge sort is a sorting algorithm that works on Divide and conquer strategy in which the array is divided into sub arrays and then combined in a sorted manner. Divide and conquer works with the concept of dividing the problems into smaller sub problems and then finding solution to each subproblem and then combining it all.

Merge sort recursively splits an array until the sub arrays are left with only one element which is the base case to apply recursion. Then recursively call merge sort function on each sub array and when the sub arrays are sorted merge them into one.

The mergesort function recursively splits the array into smaller sub arrays until only one o element is left such that the base case is achieved. After that merge function recursively sorts the subarrays and combines them all.

**HOW MERGE SORT WORKS SERIALLY:**

By serial means, that one task will be executed at one time and all tasks will be executed in a sequential manner by the processor. The algorithm remains the same for sequential execution which is to split the array into multiple subarrays until the base case is achieved and then sorting the sub arrays and finally combining them all.

**MERGE SORT SERIAL EXECUTION:**

When implemented merge sort algorithm serially on 100 million and 500 million data set the following results are achieved:

Array Size: 100000000, Time Taken: 62.18 sec

Array Size: 500000000, Time Taken: 135.62 sec

**HOW MERGE SORT WORKS PARALLELY WITH OPENMP:**

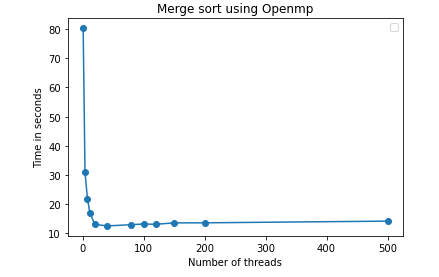
Merge sort parallel implementation with OPENMP is to take optimize the time taken to sort large data set in less time and reduce the workload on the processor when working serially. Working parallel with Openmp will be to take advantage of multiple threads to execute the merge calls.

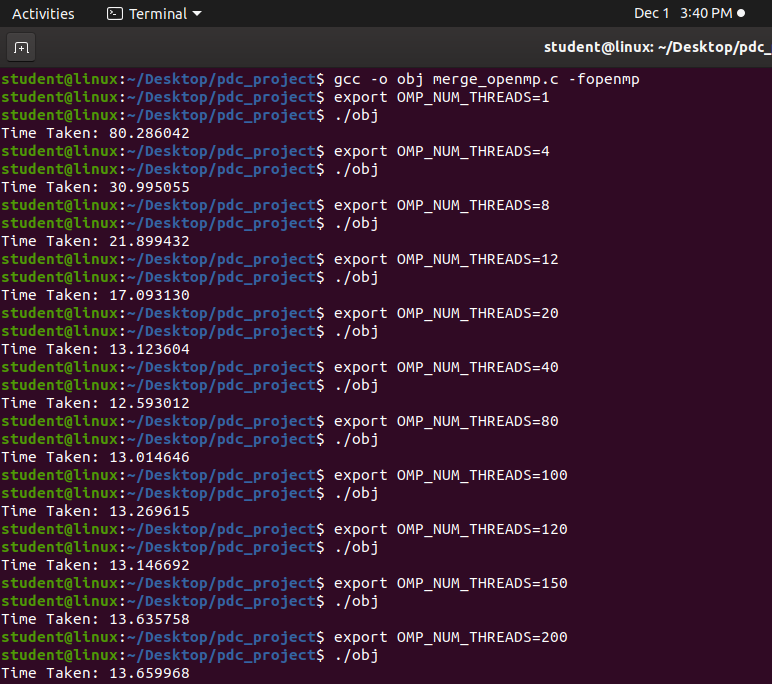
The pragmas used in the Openmp version of merge sort are:

* #pragma omp parallel: this pragma indicates the beginning of a parallel region that is all threads execute the code within this pragma in parallel.
* #pragma omp single: this pragma is used to indicated that the code written within this particular pragma will be executed by only one thread. All other threads will wait right after the single pragma barrier and complete the barrier synchronization.
* #pragma omp task: this pragma is used to indicate and define a task. The task pragma is useful for parallelizing algorithms using pointers or recursion. We can use the task pragma when we want a block of code to be executed in parallel with the code outside the task region.
* #pragma omp taskwait: this pragma is used to determine a wait on the completion of child tasks of the current task.
* firstprivate: this clause declares the scope of the data variables in list private to each thread.

**MERGE SORT OPENMP EXECUTION:**

**100 million:**





**HOW MERGE SORT WORKS PARALLELY WITH MPI:**

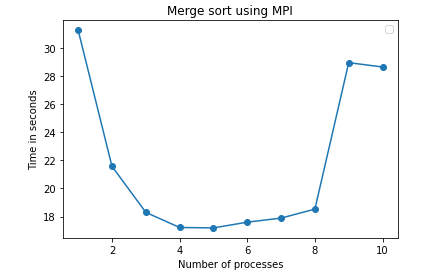
Merge sort parallel implementation using MPI is to set up many processes that allows communication as message passing paradigm. Using MPI merging of sublists is done in parallel between processes.

The MPI constructs used in the code are as follows:

* MPI\_Comm\_rank: to determine the process id for each process.
* MPI\_Comm\_size: to determine the number of processes.
* MPI\_Scatter: to scatter elements of the array to each process in a communicator. A designated root process sends data to all other processes.
* MPI\_Gather: to gather results from different processes to a single process.
* MPI\_Barrier: to determine synchronization. MPI\_Barrier blocks the caller until all processes have called it.

**MERGE SORT MPI EXECUTION:**

**100 million:**



**OBSERVATIONS:**

**ANALYSIS OF QUICK SORT USING SERIAL AND PARALLEL PROGRAMMING:**

**WHAT IS QUICK SORT:**

Quick sort is a sorting algorithm which works by breaking an array into smaller arrays and swapping the smaller ones by making comparison with the pivot element. Quick sort also works on Divide and Conquer concept. A large array is divided into two arrays one containing values smaller than the pivot and other contains values greater than pivot.

**HOW QUICK SORT WORKS SERIALLY:**

By serial means, that one task will be executed at one time and all tasks will be executed in a sequential manner by the processor. The algorithm remains the same for sequential execution which is to split the array into multiple subarrays and making comparisons of the sub arrays based on the pivot value.

Picking any element as pivot and two variables that will point left (lower index) and right (higher index) of the array excluding pivot. Moving all elements greater than pivot to the right and smaller elements to the left.

**QUICK SORT SERIAL EXECUTION:**

When implemented quick sort algorithm serially on 100 million and 500 million data set the following results are achieved:

Array Size: 100000000, Time Taken: 207.09 sec

Array Size: 500000000, Time Taken: 4817.70 sec

**HOW QUICK SORT WORKS PARALLELY WITH OPENMP:**

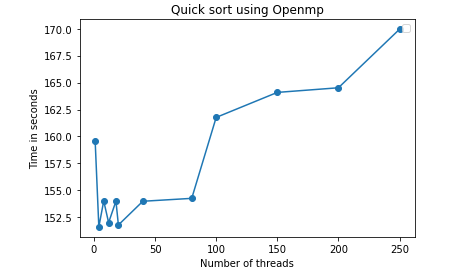
Quick sort parallel implementation with OPENMP is to take optimize the time taken to sort large data set in less time and reduce the workload on the processor when working serially. Working parallel with Openmp, for each sub array we assign a set of portions of data to each thread.

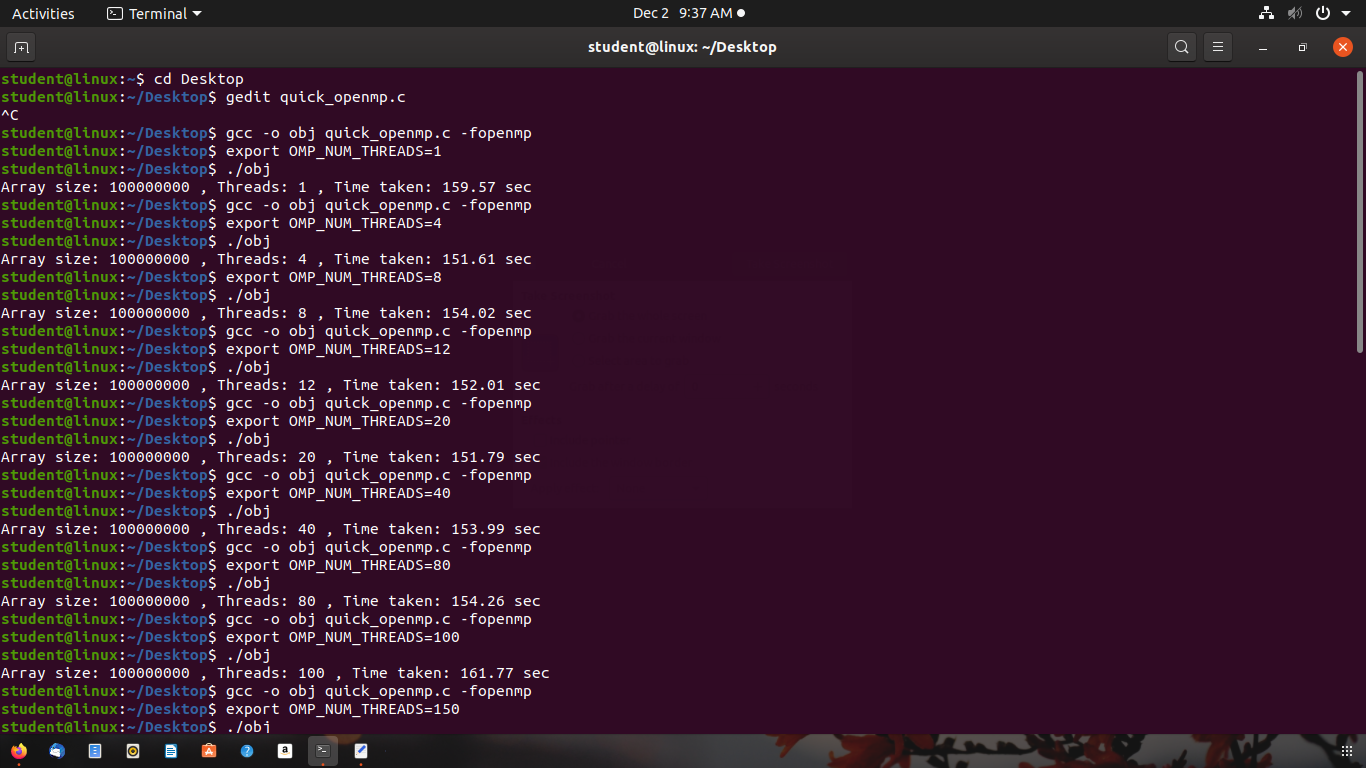
The pragmas used in the Openmp version of merge sort are:

* #pragma omp parallel: this pragma indicates the beginning of a parallel region that is all threads execute the code within this pragma in parallel.
* #pragma omp single: this pragma is used to indicated that the code written within this particular pragma will be executed by only one thread. All other threads will wait right after the single pragma barrier and complete the barrier synchronization.
* #pragma omp section: section construct is a way to distribute different tasks to different threads. Each section defines a different block which represents different tasks that is to executed by different threads.

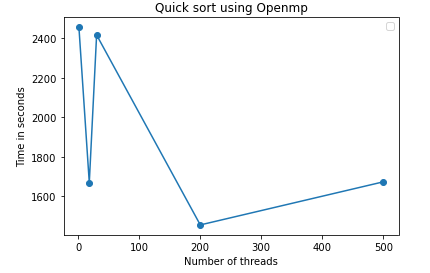
**QUICK SORT OPENMP EXECUTION**:

**100 million:**





**500 million:**



**HOW QUICK SORT WORKS PARALLELY WITH MPI:**

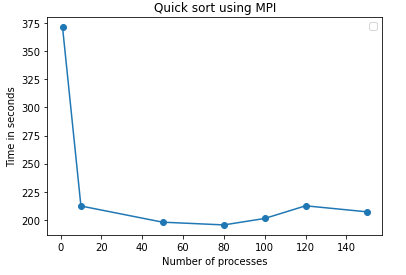
Quick sort parallel implementation using MPI is to set up many processes that allows communication as message passing paradigm. Using MPI merging of sublists is done in parallel between processes.

The MPI constructs used in the code are as follows:

* MPI\_Comm\_rank: to determine the process id for each process.
* MPI\_Comm\_size: to determine the number of processes.
* MPI\_Scatter: to scatter elements of the array to each process in a communicator. A designated root process sends data to all other processes.
* MPI\_Gather: to gather results from different processes to a single process.
* MPI\_Barrier: to determine synchronization. MPI\_Barrier blocks the caller until all processes have called it.

**QUICK SORT MPI EXECUTION:**

**100 million:**



**OBSERVATIONS:**

**REFERENCES:**

<https://www.sjsu.edu/people/robert.chun/courses/cs159/s3/T.pdf>

<https://gist.github.com/vnkdj5/0471d4ff02371eb5bb0a8773127a448d>

<https://stackoverflow.com/questions/13811114/parallel-merge-sort-in-openmp>

<https://www.geeksforgeeks.org/sorting-algorithms/>

<https://github.com/koszio/QuickSort-with-OpenMP#:~:text=The%20quicksort%20algorithm%20sorts%20the,other%20numbers%20are%20compared%20to>.

**CONCLUSION:**

Concluding it all in a nutshell, sorting algorithms are required to solve complex problems. Different sorting algorithms can be used depending on the problem requirements. We can implement these sorting algorithms serially but what to do when dealing with a huge amount of data and the problem to be solved is complex? Then obviously we would be using some parallel techniques. But make sure to analyze the most optimal time on which the threads or processes performance is the most ideal for the particular problem. Not necessarily the increase in threads or processes also increase the optimal time of execution because the communication cost might increase. Hence, it must be made sure that proper analysis is being done while implementing the desired problem through parallel techniques.