BAISONI GROUP

* Aim : Write a program to implement Parallel Bubble Sort and Merge Sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithm.

* Objective : 1) To learn parallel bubble sort & merge sort using OpenMP.
2) To measure Sequential & Parallel algorithms.

* Problem Statement : Measure the performance of Sequential and parallel algorithms.

* S/w and H/w requirements :
- 64-bit operating System
- Java, C++, python programming languages
- Jupyter Notebook.

* Theory :

Parallel Bubble Sort :

Parallel bubble Sort is a parallel implementation of the classic bubble Sort algorithm. The concept of parallelism involves executing a set of instructions code simultaneously instead of line by line sequentially (the traditional way the

* Algorithm :

i) For k = 0 to n-1
ii) If k is even then
iii) for i = 0 to (n/2) do in parallel
iv) If A[2i] > A[2i+1] then
v) Exchange A[2i] ↔ A[2i+1]
vi) Else
vii) for i = 0 to (n/2)-1 do in parallel
viii) If A[2i+1] > A[2i+2] then
ix) Exchange A[2i+1] ↔ A[2i+2]
x) Next k.

code is executed in most machines and computer programming generally.

OpenMP is a widely adopted shared memory parallel programming interface providing high level programming construct that enable the user to easily expose an application's task and loop level parallelism in an incremental.

The implementation of multithreading, a parallel execution scheme where the master thread assigns a specific number of threads to slaves the threads and a task divided between them.

Bubble Sort working:

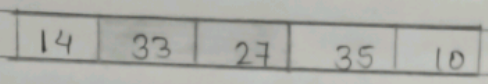
We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

14	33	27	35	10

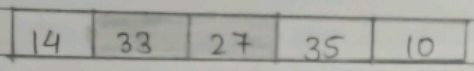
Bubble sort starts with very first two elements, comparing them to check which one is greater.

14	33	27	35	10

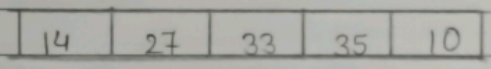
In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.



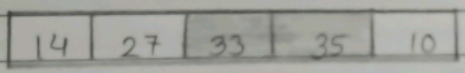
We find that 27 is smaller than 33 & these two values must be swapped.



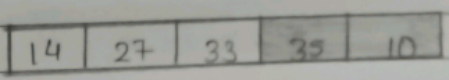
The new array should look like this -



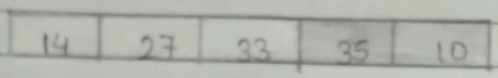
Next we compare 33 and 35. We find that both are in already sorted positions.



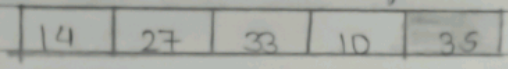
Then we need move to the next two values, 35 and 10.



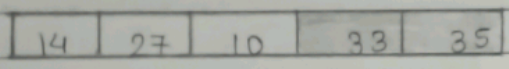
We know then that 10 is smaller 35. Hence they are not sorted.



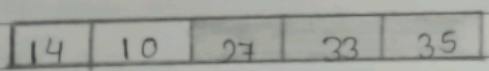
We swap these values. we find that we have reached the end of the array.



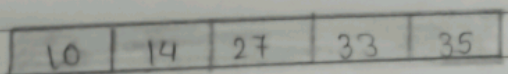
To be precise, we are now showing how an array should look like after each iteration.



Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that an array is completely sorted



Now, we should look into some practical aspects of bubble Sort.

* Algorithm of Merge Sort :

i) if it is only one element in the list it is already sorted, return.
ii) divide the list recursive into two halves until it can no more divided.
iii) merge the smaller lists into new list in sorted order.

RAISONI GROUP

* Merge Sort :

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O (n log n) it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and the combines them in a sorted manner.

* Merge Sort working :

14	33	27	10	35	19	42	44

We know that merge sort first divide the whole array iteratively into equals halves unless the atomic values are achieved.

14	33	27	10		35	19	42	44

This does not change the sequence of appearance of item in the original.

14	33		27	10		35	19		42	44

We further divide these arrays and we achieve atomic values which can no more be divided.

14	33	27	10	35	19	42	44

Now, we combine them in exactly the same manner as they were broken down. We first compare the element for each list and then combine them into a list in a sorted manner. We see that 14 & 33 are in sorted positions. We compare 27 and 10 in the target list of 2 values we put 10 first, followed by 27 we change the order of 19 and 35 where 42 & 44 are placed sequentially.

14	33	10	27	19	35	42	44

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

10	14	27	23	19	33	42	44

After the final merging, the list should look like this.

10	14	19	27	33	35	42	44

* Conclusion : Here we studied parallel sort & merge sort for using OpenMP & execute sequential and parallel algorithm.