PAGE NO.:

DATE : / /

Assignment No. A3

Problem Statement: Parallel Sorting algorithms. For Bubble Sort and Merge Sort based on existing sequential algorithms, design and implement parallel algorithm utilizing all resources available.

Learning Objectives:
1) To understand parallel bubble sort.
2) To understand parallel merge sort.

Learning Outcomes:
Understand parallel bubble sort and merge sort.

Requirements:
64 bit O.S. Linux System
4 GB RAM
Eclipse IDE

Theory:

• **Bubble Sort:-**

There are two phases in this algorithm called as Odd-Even phases. In this algorithm, 'n' elements are sorted in n phases where n is even.

Consider a sequence to be sorted is <a1, a2, a3, ..., an.

PAGE NO.:

DATE : / /

The odd phase works on this basis that the elements with odd indices are compared with their neighbours are and found as out of sequence they are exchanged.

This means the pair with odd indices and their neighbours are compare exchanged for example the pairs (a_1, a_2), (a_3, a_4), ... (a_{n-1}, a_n) are compared and exchanged if not in proper sequence assume n as even here.

It happens in similar fashion consider case with even phase. In this phase elements with even indices are compare with their right neighbours

After comparison among a pair and if found as out of sequence they are exchanged. This means the pairs (a_2, a_3), (a_4, a_5) ..... (a_{n-2}, a_{n-1}). are compared and exchanged.

The sequence is sorted after performing n-phases of odd-even exchanges.

<u>Algorithm:</u>

```
Algorithm even-odd(n)
{
    for(i:=1 ; i<=n; i++)
    {
        if(i%2!=0)        // odd phase
        {
            for(j=0 ; j<= (n/2-1))
                    perfoom_exchange (a2j+1, a2j+2)
        }
        else      // even phase
        {
            for(j=0; j<=(n/2-1))
                    perfom _exchang( a2j, a2j+1);
        }
    }
}
```

Example:-

Step	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	4 ⟷	2	7 ⟷	8	5 ⟷	1	3 ⟷	6
1	2	4 ⟷	7	8 ⟷	1	5 ⟷	3	6
2	2 ⟷	4	7 ⟷	1	8 ⟷	3	5 ⟷	6
3	2	4 ⟷	1	7 ⟷	3	8 ⟷	5	6
4	2 ⟷	1	4 ⟷	3	7 ⟷	5	8 ⟷	6
5	1	2 ⟷	3	4 ⟷	5	7 ⟷	6	8
6	1 ⟷	2	3 ⟷	4	5 ⟷	6	7 ⟷	8
7	1.	2 ⟷	3	4 ⟷	5.	6 ⟷	7	8

Merge Sort:

Merge sort first divides the unsorted list into smallest possible sub-lists, compares it with the adjacent list.

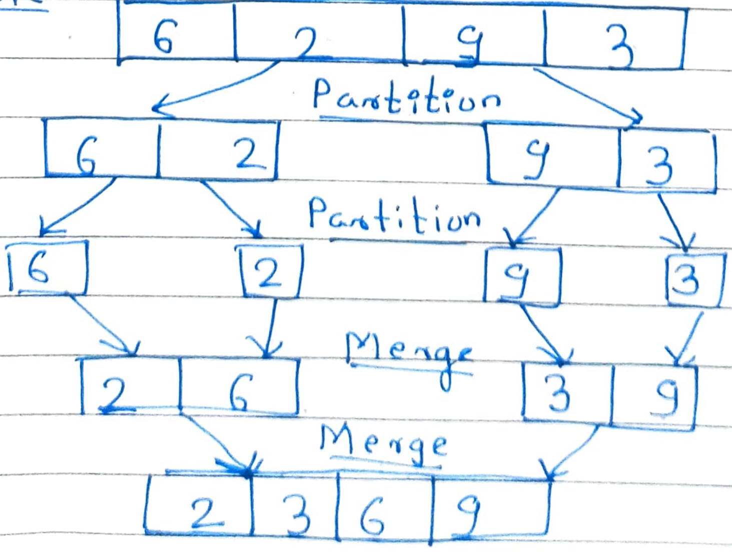
PAGE NO.:
DATE : / /

It implements parallelism very properly by following divide and conquer algorithm.

Algorithm:

```
1. mid = size/2
2. if both children present in tree then
    2.1. send mid, firstchild
    2.2. send size_mid, secondchild
    2.3 send li~~m~~st_mid, firstchild
    2.4 send list from mid, size_mid, secondchild
    2.5 call_merge( list, 0, mid, list, mid+1, size,
                    temp, 0, size)
    2.6 Store temp in another array list.
3. else
    3.1 call parallelMergeSort(list, 0, size)
4. if i>0 then
    4.1 send list, size, parent.
```

Example:



PAGE NO.:
DATE : / /

Test Cases and Analysis:

Sorting	Input Size	Sequential Time	Parallel Time	Efficiency
Bubble Sort	n=256	0.02	0.05	0.4
	n=1024	0.07	0.011	0.36
	n=2048	0.037	0.018	2.04
Merge Sort	n=256	0.001	0.02	0.05
	n=1024	0.003	0.03	0.1
	n=2048	0.002	0.02	0.1

Efficiency : WCSA / WCPA

```
I/P:  5  9  5  5  24  11  15  5  0

O/P:  0  5  5  5  5  9  11  15  24
```

Commands to run:

```
gcc -fopenmp srcfile.cpp -o outputfile
```

Conclusion:

Thus, we successfully implemented parallel bubble sort and merge sort using open mp.