Assignment No: Group A_02

Aim:

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

Objective:

Student will learn:

i) The Basic Concepts of Bubble Sort and Merge Sort.

ii) Multiple Compiler Directives, library routines, environment variables available for OpenMP.

Theory:

Introduction:

Parallel Sorting:

A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore, parallel algorithms are used in sorting.

Design methodology:

Based on an existing sequential sort algorithm

- Try to utilize all resources available
- Possible to turn a poor sequential algorithm into a reasonable parallel algorithm

Bubble Sort

The idea of bubble sort is to compare two adjacent elements. If they are not in the right order, switch them. Do this comparing and switching (if necessary) until the end of the array is reached. Repeat this process from the beginning of the array n times. Average performance is $O(n^2)$

Bubble Sort Example

Here we want to sort an array containing [8, 5, 1].

8, 5, 1	Switch 8 and 5
5, 8, 1	Switch 8 and 1
5, 1, 8	Reached end start again.
5, 1, 8	Switch 5 and 1
1, 5, 8	No Switch for 5 and 8
1, 5, 8	Reached end start again.
1, 5, 8	No switch for 1, 5
1, 5, 8	No switch for 5, 8
1, 5, 8	Reached end.

But do not start again since this is the nth iteration of same process

Parallel Bubble Sort

• Implemented as a pipeline.

• Let local_size = n / no_proc. We divide the array in no_proc parts, and each process executes the bubble sort on its part, including comparing the last element with the first one belonging to the next thread.

• Implement with the loop (instead of j<i) for (j=0; j<n-1; j++)

• For every iteration of i, each thread needs to wait until the previous thread has finished that iteration before starting.

• We'll coordinate using a barrier.

Algorithm for Parallel Bubble Sort

1. For k = 0 to n-2
2. If k is even then
3. for i = 0 to (n/2)-1 do in parallel
4. If A[2i] > A[2i+1] then

```
5.    Exchange A[2i] ↔ A[2i+1]
6.    Else
7.    for i = 0 to (n/2)-2 do in parallel
8.    If A[2i+1] > A[2i+2] then
9.    Exchange A[2i+1] ↔ A[2i+2]
10.   Next k
```

Parallel Bubble Sort Example

• Compare all pairs in the list in parallel

• Alternate between odd and even phases

• Shared flag, sorted, initialized to true at beginning of each iteration (2 phases), if any processor perform swap, sorted = false

Merge Sort

- Collects sorted list onto one processor
- Merges elements as they come together
- Simple tree structure
- Parallelism is limited when near the root

Steps of Merge Sort:

To sort A[p .. r]:

1. Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split A[p .. r] into two subarrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r]. That is, q is the halfway point of A[p .. r].

2. Conquer Step

Conquer by recursively sorting the two subarrays A[p .. q] and A[q + 1 .. r].

3. Combine Step

Combine the elements back in A[p .. r] by merging the two sorted subarraysA[p .. q] and A[q + 1 .. r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Example:

Parallel Merge Sort

- Parallelize processing of sub-problems
- Max parallelization achived with one processor per node (at each layer/height)

Parallel Merge Sort Example

- Perform Merge Sort on the following list of elements. Given 2 processors, P0 & P1.
- 4,3,2,1

Algorithm for Parallel Merge Sort

1. Procedure parallelMergeSort
2. Begin
3. Create processors Pi where i = 1 to n
4. if i > 0 then recieve size and parent from the root
5. recieve the list, size and parent from the root
6. endif
7. midvalue= listsize/2
8. if both children is present in the tree then
9. send midvalue, first child
10. send listsize-mid,second child

```
11.  send list, midvalue, first child
12.  send list from midvalue, listsize-midvalue, second child
13.  call mergelist(list,0,midvalue,list, midvalue+1,listsize,temp,0,listsize)
14.  store temp in another array list2
15.  else
16.  call parallelMergeSort(list,0,listsize)
17.  endif
18.  if i >0 then
19.  send list, listsize,parent
20.  endif
21.  end
```

ALGORITHM ANALYSIS

1. Time Complexity Of parallel Merge Sort and parallel Bubble sort in best case is(when all data is already in sorted form):**O(n)**

2. Time Complexity Of parallel Merge Sort and parallel Bubble sort in worst case is: **O(n logn)**

3. Time Complexity Of parallel Merge Sort and parallel Bubble sort in average case is: **O(nlogn)**

Conclusion: Thus, we have successfully implemented parallel algorithms for Bubble Sort and Merger Sort.