High Performance Computing : Assignment 2

Parallel sorting

Oral questions

1. What is the Parallel formulation of odd even transposition sort.

It is based on the Bubble Sort technique, which compares every 2 consecutive numbers in the array and swap them if first is greater than the second to get an ascending order array. It consists of 2 phases – the odd phase and even phase:

- **Odd phase:** Every odd indexed element is compared with the next even indexed element(considering 1-based indexing).
- **Even phase:** Every even indexed element is compared with the next odd indexed element.

2. What is the Parallel formulation of merge sort.

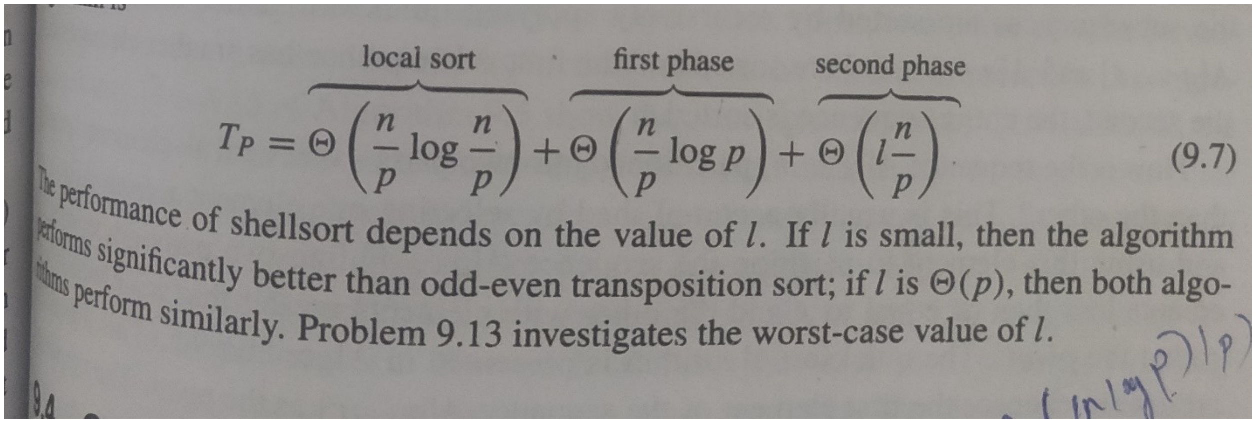
   Merge sort first divides the unsorted list into smallest possible sub-lists, compares it with the adjacent list, and merges it in a sorted order. It implements parallelism very nicely by following the divide and conquer algorithm.

3. Which decomposition method is used in parallelizing merge sort

   Divide & Conquer

4. Explain parallel formulation of odd even transposition sort on an n processing ring

5. State parallel run time of shell sort.



$$T_P = \underbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}_{\text{local sort}} + \underbrace{\Theta\left(\frac{n}{p}\log p\right)}_{\text{first phase}} + \underbrace{\Theta\left(l\frac{n}{p}\right)}_{\text{second phase}} \qquad (9.7)$$

The performance of shellsort depends on the value of l. If l is small, then the algorithm performs significantly better than odd-even transposition sort; if l is $\Theta(p)$, then both algorithms perform similarly. Problem 9.13 investigates the worst-case value of l.

(n log p)/p)

6. Compare sequential and parallel merge sort and odd even transposition sort.

- Sequential bubble sort, O(n^2)

  The time complexity is reduced to O(N) due to parallel computation using threads.

- Merge sort, O(nlogn)

In parallel, n processors are present.

logn time is required to divide the sequence and logn time is required to merge.

Hence, logn+logn=2logn

The time complexity is reduced to O(logn) due to parallel computation using threads.

7. How to parallelise quicksort? State methods.

   Shared-address

   Message passing

8. Commend on pivot selection.

9.4.4 Pivot Selection

In the parallel quicksort algorithm, we glossed over pivot selection. Pivot selection is particularly difficult, and it significantly affects the algorithm's performance. Consider the case in which the first pivot happens to be the largest element in the sequence. In this case, after the first split, one of the processes will be assigned only one element, and the remaining $p - 1$ processes will be assigned $n - 1$ elements. Hence, we are faced with a problem whose size has been reduced only by one element but only $p - 1$ processes will participate in the sorting operation. Although this is a contrived example, it illustrates a significant problem with parallelizing the quicksort algorithm. Ideally, the split should be done such that each partition has a non-trivial fraction of the original array.

One way to select pivots is to choose them at random as follows. During the i^{th} split, one process in each of the process groups randomly selects one of its elements to be the pivot for this partition. This is analogous to the random pivot selection in the sequential quicksort algorithm. Although this method seems to work for sequential quicksort, it is not well suited to the parallel formulation. To see this, consider the case in which a bad pivot is selected at some point. In sequential quicksort, this leads to a partitioning in which one subsequence is significantly larger than the other. If all subsequent pivot selections are good, one poor pivot will increase the overall work by at most an amount equal to the length of the subsequence; thus, it will not significantly degrade the performance of sequential quicksort. In the parallel formulation, however, one poor pivot may lead to a partitioning in which a process becomes idle, and that will persist throughout the execution of the algorithm.

If the initial distribution of elements in each process is uniform, then a better pivot selection method can be derived. In this case, the n/p elements initially stored at each process form a representative sample of all n elements. In other words, the median of each n/p-element subsequence is very close to the median of the entire n-element sequence. Why is this a good pivot selection scheme under the assumption of identical initial distributions? Since the distribution of elements on each process is the same as the overall distribution of the n elements, the median selected to be the pivot during the first step is a good approximation of the overall median. Since the selected pivot is very close to the overall median, roughly half of the elements in each process are smaller and the other half larger than the pivot. Therefore, the first split leads to two partitions, such that each of them has roughly $n/2$ elements. Similarly, the elements assigned to each process of the group that is responsible for sorting the smaller-than-the-pivot elements (and the group responsible for sorting the larger-than-the-pivot elements) have the same distribution as the $n/2$ smaller (or larger) elements of the original list. Thus, the split not only maintains load balance but also preserves the assumption of uniform element distribution in the process group. Therefore, the application of the same pivot selection scheme to the sub-groups of processes continues to yield good pivot selection.

Can we really assume that the n/p elements in each process have the same distribution as the overall sequence? The answer depends on the application. In some applications,

9. Which algorithmic strategy is used in merge sort?

   Divide and Conquer

10. What is parallel runtime of matrix vector multiplication

11. Is it possible to implement parallel sorting using a cluster environment?

Practice problem

- Implement parallel sorting using cluster of Raspberry Pi
- Check the performance of this program by varying the number of nodes in a cluster and plot the graph.
- Implement parallel shell sort.
- Implement parallel quick sort using shared address space platform.
- Implement radix sort.
- Implement any sorting algorithm

  1. Take a input from file
  2. Use Random function
  3. Provide a input at run time

Similarly print output in afile