ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

Advanced Quick Sort with Parallel Processing

Dixit Bali[1], Anshu Garg[2], Madhusudan[3]

Student, School of Computer Science & Engineering, Bahra University, Shimla, H.P, India[1,2]

Assistant Professor, School of Computer Science & Engineering, Bahra University, Shimla, H.P, India[3]

Abstract: Quick sort algorithm is a sorting algorithm that works on the principle of divide-and-conquer, making sub-lists(sub-array) out of the given problem domain and then recursively applying the same swapping technique till the whole list is sorted. There are numerous operations involving comparison, swapping and assignments in the quick sort algorithm. In this paper we have proposed a new algorithm that is an enhanced version of original quick sort. The idea behind this is to reduce the number of steps, in order to achieve that we have introduced parallel processing which simultaneously deals with number of sub-arrays at the same time. The algorithm is recursive, but with the help of parallel processing the number of steps is considerably reduced and all the cases can be easily dealt with. There is more requirement of memory in this algorithm since all the sub-arrays are parallel processed, but there is significant reduction in time complexity when the list contains large number of elements.

Keywords: Slots, parallel processing, touched elements, reference element, MIN and MAX.

I. INTRODUCTION

Sorting is a technique used in the field of computer science to rearrange a given list of elements in a specified logical order [1], for example for character input, rearranging the list in an increasing order of alphabets. Sorting is one the most important task that has to be performed efficiently when we have to work with big data [2].The entire Database based applications need sorting in a way that saves time and increases visual consistency of the data [3]. Certain algorithms are devised to carry out the process of Sorting. According to [5], one of the most efficient algorithms to Sort, as developed by [4], is Quick sort algorithm. Quick sort is the easiest of all the sorting algorithms, and has optimum resource utilization[6]. It is a divide and conquer algorithm, which means, the full list is divided into sub-lists and this process is recursive till we find the correct position of all the elements [7]. The elements are rearranged in an increasing order of their existence. The elements in the left sub-array are less than the pivot element and the right sub-array has all the elements greater than the pivot. Thus the pivot is the basis of comparison between left and right sub-array. The boundary elements are stored in the stack namely the LOWER and UPPER stack respectively [8]. Thus the Push and Pop operations are applied on these stacks.

II. ADVANCED QUICK SORT: AN IDEA

Advanced Quick sort is the advanced implementation of the original Quick sort algorithm given by [9]. Though the basic working is the same, that is, in this proposed algorithm we are using the same sub-list concept. The traversing direction is changing after every step as it was changing in the case of [10], starting from right to left traversal. The list is traversed from right to left and left to right respectively in every adjacent step so as to find the least or the greatest element with respect to the reference element in every step and hence interchanging the location of both elements.

In this proposed algorithm, the list is divided into sub units/sub-lists called slots. The slot starts from the first touched element to the next touched element. These slots are parallel processed and in one step, all the slots are processed and executed. If there is interdependence between two elements to interchange to a single slot position, we also have mentioned a special case scenario to carry out the process smoothly without any error. The reference element in the starting of the process is the extreme left element in every slot (this is the case when the list is traversed from right to left) otherwise the right most element of the slot. Every element that is interchanged is highlighted and thus it becomes the reference element for its own slot.

Before carrying out the advanced Quick sort technique, we find the MIN and MAX elements in the list and interchange them with the A[K] and A[N] respectively in the array of A[K],A[K+1],…,A[N]. This sorts the first and last element of the array. Now we can carry the proposed technique.

In this proposed algorithm, we have implemented a technique called Parallel Processing. This is a method of simultaneously solving each slot in parallel in a given step. This helps us to improve the efficiency of [11].

III. EXAMPLE OF ADVANCED QUICK SORT

Question :- Sort the list:
44,33,11,55,77,90,40,60,99,22,88,66.

STEP 1 (A)

Find the location LOC of the smallest element in the list of N elements A[1],A[2],A[3],….,A[N], and then interchange A[LOC] and A[1]. Then A[1] is sorted.

Set MIN: = A[K] and LOC: = K and then traverse the list, comparing MIN with each other element A[J] in the array from A[K] to A[N].

a) If MIN <= A[J], then simply move to the next element.
b) If MIN >A[J], then update MIN and LOC by setting MIN := A[J] and LOC:= J.

After comparing MIN with the last element A[N], MIN will contain the smallest among the elements among A[K],A[K+1], …,A[N] and LOC will contain its location. [12]
The smallest element is 11 at A[3] so interchange A[1] A[3] , that is, interchange 44 and 11 to get the list:
11,33,44,55,77,90,40,60,99,22,88,66

STEP 1 (B)
Similarly, find the largest element in the list and interchange the largest element with A[N] element in the array A[K],A[K+1],…,A[N]. Thus A[N] will be sorted.
Set MAX: = A[K] and LOC: = K and then traverse the list, comparing MAX with each other element A[J] in the array from A[K] to A[N].

a) If MAX >= A[J], then simply move to the next element.
b) If MAX <A[J], then update MAX and LOC by setting MAX := A[J] and LOC:= J.

After comparing MAX with the last element A[N], MAX will contain the largest among the elements A[K], A[K+1],…,A[N] and LOC will contain its location.
The largest element is 99 at A[9] so A[9] □□A[12], that is, interchange 66 and 99 to get the list:
11,33,44,55,77,90,40,60,66,22,88,99

STEP 2
Now we will consider the output list of Step 1 b).
As 11,44,66,99 are interchanged/touched elements, we will consider them as the basis of comparison.

Use the first touched element that is 11 in the list at LOC: = 1 and so reference element REF: = A[1] and consider the length of the slot till the next touched element that is 44 at LOC: = 3.

Now we check any element less than 11 traversing the list from right to left from 44(44 included).

Thus we end up meeting 11 as no element is less than 11.
So now we continue the process and thus now the reference element REF: = A[3] at LOC: = 3, which is element 44 and the slot is narrowed down from 44 till 66(44 and 66 included) and traversed from right to left only, in the same step, finding an element less than 44.

Thus from 66, 40 is the element that is less than REF and so we interchange A[3] □□A[7].

And lastly in the same step we consider the slot 66 to 99 with reference element REF: = A[9] and thus search an element less than 66 traversing from right to left of this individual slot.

We get 22 so we interchange A[9] and A[10].
Thus combining all the slots, we get the list:
11,33,40,55,77,90,44,60,22,66,88,99

STEP 3
Now again we check the list considering interchanged/touched elements but this time changing the traversing direction from left to right finding the greatest in each slot.

Thus the first slot is from 11 to 40 with the reference element to be the right most in the slot REF: = A[3], that is, element 40 and traversing from left to right that is, from 11 finding an element greater than 40.
We do not meet any element before meeting the reference element REF itself.

So we continue the sorting and go to the next slot of elements lying between touched elements 40 and 44(both included). This time the reference element is the rightmost in the slot that is element 44. So we traverse from 40 and find an element greater than 44, which is 55 at A[4] so A[4]□□A[7].

Now we move onto the next slot that is from 44 to 22. REF: = A[9] which is element 22 and we need to find an element greater than 22.

44 is the element greater than 22. Thus A[7] □□A[9].
But A[7] is being interchanged with A[4] also.
This leaves us with one of the few special cases:
An element being used in two slots that is being interchanged at two places.

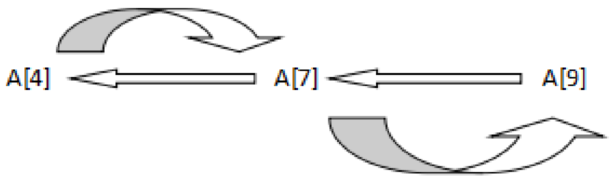

Fig. 1. Swapping in Exceptional Case

Thus we apply one special rule. We keep the middle element constant and interchange the both conflicting extreme elements that is A[4] □□ A[9] leaving A[7] untouched.

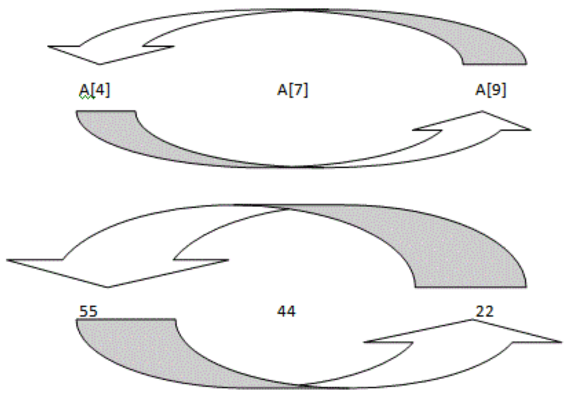

Fig. 2. Exchanging Only Touched Elements

Next we move to the slot between 22 and 66(both included). Thus the REF: = A[10] that is element 66 and we find the greatest element traversing from left to right thus getting no element reaching the REF itself. So we move onto the next slot.

Now we consider 66 and 99 and reference element is 99 and we know this is the Max of the list thus no element greater than 99 in the slot.
The list after this step is:
11,33,40,22,77,90,44,60,55,66,88,99

STEP 4
Now we will consider the above list of Step 3 and will check the least element traversing from right to left in every slot.

Thus first slot is 11 to 40(both included). Reference element is 11 at LOC: = 1 and we will traverse from right to left that is from 40 to 11 finding the least element than 11. There is no element meeting the requirement so we move to the next slot.

The next slot is from 40 to 22. Reference element is 40 and we find an element less than 40 which in turn is 22. Interchange 40 and 22. Move to the next slot.

The next slot is from 22 to 44. Reference element is 22 and we find an element traversing from 44 to 22 that is less than 22. There is no element like that so we move to the next slot.

The next slot is from 44 to 55. Reference element is 44 and there is no element less than 44 when slot is traversed from right to left starting from 55 so we move to the next slot.

The next slot is from 55 to 66 and there is no element less than 55 when traversed form 66 towards 55. So we move to the next slot.

The next slot is from 66 to 99 of the touched elements on the extreme end and thus we find a number less than 66 from 99 but no element satisfies the condition.
So the list now is:
11,33,22,40,77,90,44,60,55,66,88,99

STEP 5
Now we will change the traversing direction from left to right in every slot and check the greatest element comparing with the right most touched element in the slot.
So the element 22□□ 33 ; 77□□ 44 ; 60□□ 55 are interchanged in different slots by the similar method.
So the list is:

11,22,33,40,44,90,77,55,60,66,88,99

STEP 6
Now we take the output list of the previous step and check the slots by considering the least element traversing from right to left with reference element to be the left untouched element of the slot.
So the element 77□□ 55 is interchanged to give a new list:
11,22,33,40,44,90,55,77,60,66,88,99

STEP 7
We traverse the list from left to right.Find the greatest element in the slot with the reference element to be the right touched element of the slot.

So the element 90□□55 ; 60□□77 will be interchanged.
So the list is:

11,22,33,40,44,55,90,60,77,66,88,99

STEP 8
Next, we traverse the list from right to left finding the least element in the slot, with the reference element to be the left touched element of the slot.
So the element 90□□ 60; 66□□ 77 will be interchanged.
So the list is:
11,22,33,40,44,55,60,90,66,77,88,99

STEP 9
In the next step we traverse the list from left to right finding the greatest element in the slot, with the reference element to be the right touched element of the slot.
So the element 90 □□ 66 is interchanged.
So the list is:
11,22,33,40,44,55,60,66,90,77,88,99

STEP 10
Now considering the above list and following the same pattern that is changing the traversing direction from right to left and changing the reference element to be the left touched element of the slot.
So the element 90 □□ 77 is interchanged.
So the list is:
11,22,33,40,44,55,60,66,77,90,88,99

STEP 11
Again the same pattern has to be followed that results in traversing the list from left to right with the reference element to be the right touched element of the slot.
Thus no element is greater than the reference element of each slot respectively. So the list remains the same.

STEP 12
Again the list has to be traversed from right to left with the reference element in each slot to be left extreme element of the individual slots and thus the interchanging is done between 90□□ 88 ,thus giving a sorted list:
11,22,33,40,44,55,60,66,77,88,90,99

IV. COMPARISON WITH QUICK SORT

We are solving the same example using quick sort (Randomized).

Let the given input string of numbers to be sorted is:44,33,11,55,77,90,40,60,99,22,86,66

{44,33,11,55,77,90,40,60,99,22,86,66} (1)
{44,33,11,55,66,90,40,60,99,22,86,77} (2)
{44,33,11,55,66,22,40,60,99,90,86,77} (3)

Now we have two arrays, so apply the above steps individually on each sub-array.

{44,33,11,55,66,22,40,60} {99,90,86,77}
{44,33,11,55,66,22,40,60} {99,90,86,77} (4)

{44,33,11,55,60,22,40,66} {77,90,86,99} (5)
{44, 33,11,55,60,22,40} 66 77 {90,86,99} (6)

{40, 33,11,55,60,22,44} 66 77 {90,86}99 (7)
{40, 33,11,22,60,55,44} 66 77 {90, 86} 99 (8)
{40,33,11,22}{60,55,44} 66 77 {90,86}99
{40,33,11,22}{60,55,44}66 77 {90,86} 99 (9)
{22,33,11,40} {44,55,60} 66 77 {86,90} 99 (10)
{22,33,11}40 44 {55,60} 66 77 {86,90} 99
{22,33,11} 40 44 {55, 60} 66 77 {86,90} 99 (11)
{11,33,22} 40 44 {55,60} 66 77 {86,90} 99 (12)
11 {33,22} 40 44 {55,60} 66 77 {86,90} 99 (13)
11 33, 22 40 44 {55,60} 66 77 {86,90} 99 (14)
11 {22,33} 40 44 {55,60} 66 77{86,90}99 (15)

Combined list is as: -
11,22,33,40,44,55,60,66,77,86,90,99
Title must be in 24 pt Regular font. Author name must be in 11 pt Regular font. Author affiliation must be in 10 pt Italic. Email address must be in 9 pt Courier Regular font.

V. RESULTS

For array1 {3,8,5,7,9,6,2,4,10,1} having ten elements.
For array2 {53,21,63,47,90,86,24} having seven elements.

No. of elements to be sorted	Steps taken by advanced quick sort	Steps taken by quick sort
12	12	15
10	11	14
07	04	06

VI. CONCLUSION

This paper proposed a new algorithm for sorting elements, which is the advanced version of quick sort, termed as Advanced Quick Sort. We have applied both the algorithms namely quick sort & advanced quick sort on same set of elements and the obtained results are listed above. The comparison was performed on the number of steps taken by both the algorithms. It has been found that the number of steps taken by advanced quick sort is comparatively less as compared to the original quick sort (randomized).

FUTURE SCOPE

Though this paper provides a clear overview about how Advanced Quick Sort is better than the original quick sort, but an algorithm has not yet been proposed on a mathematical model. This could be an area for further advances in this field. Also there are some special cases that cannot yet be solved using the above mentioned steps, so a better algorithm can be proposed to deal with such type of exceptional cases.

REFERENCES

[1] Robert Sedgewick, "Implementing Quick sort Programs", Division of Applied Mathematics and Computer Science Program, Brown University, Providence, RI 02912, 1978 ACM 0001-0782/78/1000-0847.
[2] Abdulrahman Hamed Almutairi and Abdulrahman Helal Alruwaili, "Improving of Quick sort Algorithm Performance by Sequential Thread or Parallel Algorithms", GJCST, 2012.
[3] Madhavi Desai and Viral Kapadiya, "Performance Study of Efficient Quick Sort and Other Sorting Algorithms for Repeated Data", 2011.
[4] C.A.R. Hoare, "Algorithm 64: Quicksort," Comm. ACM 4, 7, 321, July 1961.
[5] Laila Khreisat, "QuickSort A Historical Perspective and Empirical Study", IJCSNS, 2007.
[6] R. Sedgewick, Algorithms in C++, 3rd edition, Addison Wesley, 1998.
[7] Seymour Lipschutz, "DATA STRUCTURES",Tata McGraw Hill,2012.
[8] Seymour Lipschutz, "DATA STRUCTURES", Tata McGraw Hill,2012.
[9] C.A.R. Hoare, "Algorithm 64: Quicksort," Comm. ACM 4, 7, 321, July 1961.
[10] C.A.R. Hoare, "Algorithm 64: Quicksort," Comm. ACM 4, 7, 321, July 1961.
[11] C.A.R. Hoare, "Algorithm 64: Quicksort," Comm. ACM 4, 7, 321, July 1961.
[12] Seymour Lipschutz, "DATA STRUCTURES", Tata McGraw Hill, 2012.