ISSN- 2394-5125 VOL 7, ISSUE 09, 2020

PARALLELIZATION OF TIM SORT ALGORITHM USING MPI AND CUDA

Siva Thanagaraja[1], Keshav Shanbhag[2], Ashwath Rao B[3][0000-0001-8528-1646], Shwetha Rai[4][0000-0002-5714-2611], and N Gopalakrishna Kini[5]

Department of Computer Science & Engineering
Manipal Institute Of Technology
Manipal Academy Of Higher Education
Manipal, Karnataka, India-576104

Abstract— Tim Sort is a sorting algorithm developed in 2002 by Tim Peters. It is one of the secure engineered algorithms, and its high-level principle includes the sequence S is divided into monotonic runs (i.e., non-increasing or non-decreasing subsequence of S), which should be sorted and should be combined pairwise according to some specific rules. To interpret and examine the merging strategy (meaning that the order in which merge and merge runs are performed) of Tim Sort, we have implemented it in MPI and CUDA environment. Finally, it can be seen the difference in the execution time between serial Tim Sort and parallel Tim sort run in O (n log n) time .

Index Terms— Hybrid algorithm, Tim sort algorithm, Parallelization, Merge sort algorithm, Insertion sort algorithm.

I. INTRODUCTION

Sorting algorithm Tim Sort [1] was designed in 2002 by Tim Peters. It is a hybrid parallel sorting algorithm that combines two different sorting algorithms. This includes insertion sort and merge sort. This algorithm is an effective method for well-defined instructions. The algorithm is concealed as a finite list for evaluating Tim sort function. Starting from an initial state and its information the guidelines define a method that when execution continues through a limited number of next states. The next states are well characterized, at the end providing output and ending at the last completion state [2].

Tim sort works by identifying runs of least two elements. Element runs occur either strictly descending (each element is lesser than its predecessor) or in ascending (each element is greater than or equal to its predecessor) order. At its worst case, it runs at a similar speed of merge sort which means it is unexpectedly very fast. In terms of space, Tim Sort is on the worse end of the spectrum, but the space consideration for most sorting algorithms is highly sparse. Concept of stability is that when array sorted, objects of same value maintain their original order. If Tim sort follows an unstable algorithm, it results in a loss of reliability from first sort when we run the second one.

The following steps for Tim sort include:

(i) The existing structure of the list is taken and n-1 operations are performed on the structure list that is either sorted or is in strictly-descending (reverse) order.

(ii) Then the algorithm scans the structure list and finds "runs" of elements that are either in strictly descending or in ascending order.

(iii) If the element runs take place in strictly descending order, reverse operation of Tim sort occurs.

(iv) If the run is less then set "min run", then the algorithm Tim sort performs Insertion Sort aggregate min run elements. Min run value is calculated based on the size of the array.

The algorithm merge runs when the value of the array exceeds the min run values and also keeps merges balanced.

II. IMPLEMENTATION

Before looking into how Tim Sort is implemented parallel, let us discuss some basics of Insertion and Merge Sort.

A. Insertion Sort & Merge Sort

Insertion Sort is one of the fundamental sorting algorithms. First it glance the array and when it finds the element which is not in proper position ,moves it to a position in array which is sorted before as shown in Fig. 1.This sorting algorithm works good in situation like already sorted arrays and smaller arrays. [3]. As we can conclude from Fig.1, that complexity of Insertion Sort is O (n) [4]. In Tim Sort when array is already sorted, insertion sort works well.

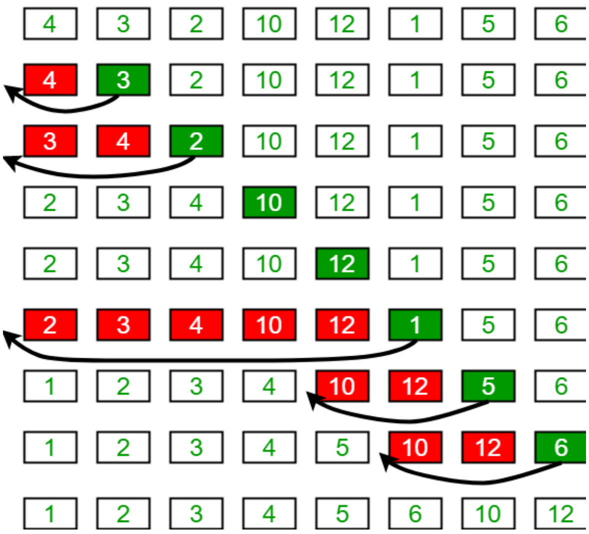

Fig 1.Insertion Sort Example (source [5])

Second Sorting algorithm is Merge Sort whose basic idea is to merge arrays that are sorted. As Divide and conquer method, it splits the array into half and again splits the two arrays into half and so on to get single element. After it starts merging single elements by sorting method as shown in Fig.2 [3]. As we started with dividing large array into single elements and again building the primary array by merging individuals, process runs in O(n log n)[4] time.

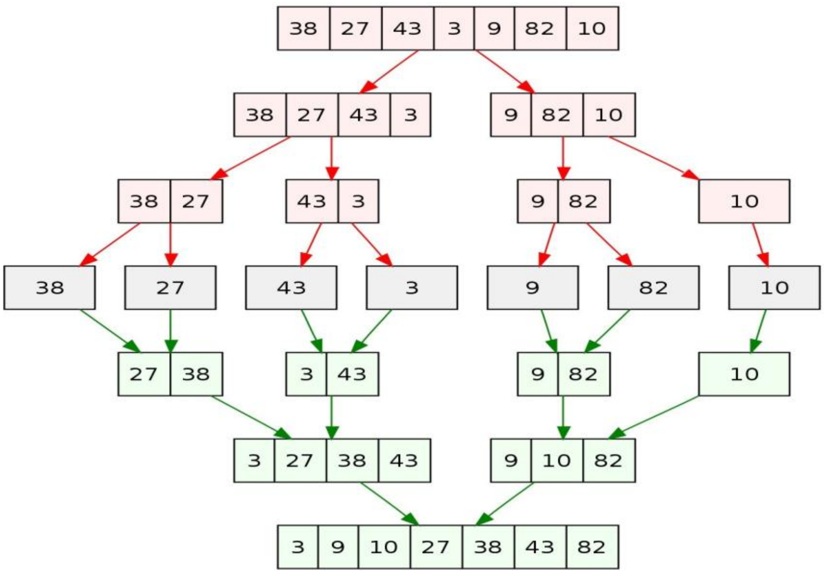

Fig.2: Merge Sort Example (Source:[6])

B. Implementing Parallel Tim Sort

The way to comprehension Tim Sort's execution understands its utilization of runs. Tim Sort influences normally happening pre-sorted information to further its potential benefit. By pre-sorted, it implies that consecutive components are on the whole expanding or diminishing. In insertion sort, one element from the input elements is consumed in each iteration to find its correct position i.e., the position to which it belongs in a sorted array. It starts by finding the current element with larger or smaller elements. Later current elements finds its suitable position by comparison. In Merge Sort, we divide an array of elements that are unsorted into sub-array of equal halves until it can no more be divided. Then merge sort combines smaller sorted lists keeping the new list sorted [7]. Note that insertion sort doesn't indulge parallelism while the merge sort can be implemented parallel.

First set a min run size. If the input to Tim sort is lesser than min run then perform insertion sort. Otherwise perform merge sort in which every sub-array obtained after divide operation in merge sort, once reaches min run size, it performs Insertion Sort as shown in Data Flow Diagram Fig 3.

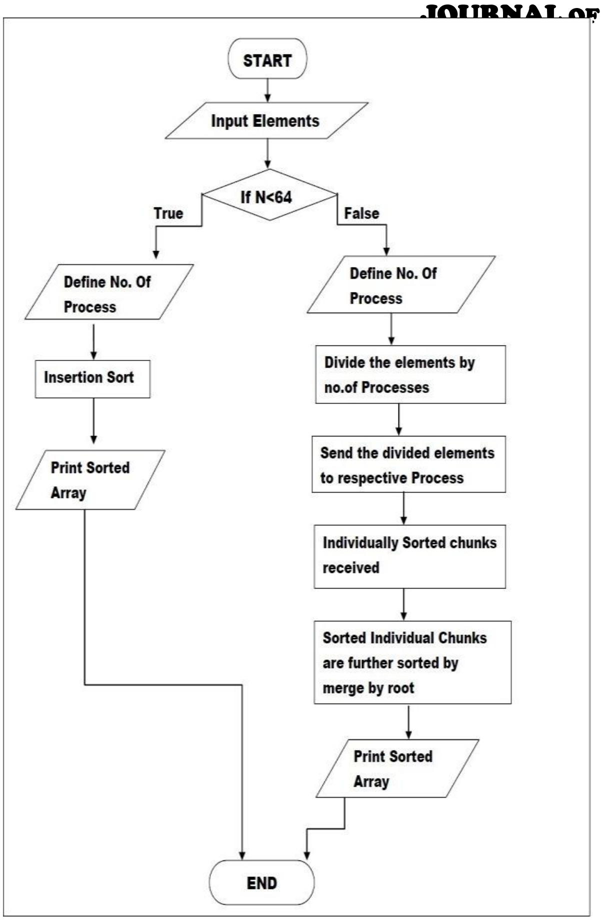

Fig.3: Data Flow Diagram of Tim Sort

Steps to implement parallel Tim sort:

a) First set up a min run size which is a power of 2(we have taken as 64 as above which insertion sort loses its efficiency) [8]

b) Find size of the array to compare with min run i.e. 64.

c) If the size is less than min run, execute insertion sort as discussed in the previous section.

.

d) If size is more than the min run then divide the entire array into size of min run and perform insertion sort to each sub arrays.

e) Perform Merge sort on each sub arrays and thus we will get entire array sorted finally.

C. For Parallelism in CUDA

In CUDA, we use threads to sort numbers as each thread controls one set of elements of size 64. First, we divide the count of total input array elements by 64, and by this, we will get a count of threads. After getting the number of threads, first, divide the entire array of size 64 for each thread to perform insertion sort where each thread perform insertion sort of their array in parallel. In the second step, groups of two threads will perform merging of their respective array elements obtained after insertion sort in parallel. So thereafter the first thread of each group will be having an array size of 128. The third step is again two threads group together to perform merging sort on their respective array of size 128 resulting in an array size of 256 held by the first thread of their group in parallel. Further, the same procedure of two threads grouping and merging technique is followed in parallel until we obtain a single array held by a single thread, which is a final sorted array.

III. ANALYSIS

When the input size is Less than equal to 64, then the time used by MPI, CUDA, and the sequential programs take the same time. In MPI with more processes are being created and in CUDA with threads the time taken by the computation DECREASES, but after a certain number of processes are already being created, then the computation time INCREASES.

The computation time is dependent on the number of processes. The number of processes must be created according to the input Size for BETTER execution time.

PARALLEL ALGORITHM ANALYSIS:

1. For input being 10 000 as the size of the array, we evaluate the time taken based on the different number of processes as shown in Table 1.

ISSN- 2394-5125 VOL 7, ISSUE 09, 2020

Table 1: For N=10 000, the number of processes v/s time.

N=10 000	
Processes	**Time**
2	**0.028327 s**
4	**0.030933 s**
8	**0.021053 s**
16	**0.041526 s**
32	**0.024638 s**
64	**0.037315 s**
128	**0.035545 s**

2. For input being 100 000 as the size of the array, we evaluate the time taken based on the different number of processes as shown in Table 2.

Table 2: For N=100000, the number of processes v/s time.

N=100 000	
Processes	**Time**
2	**0.240311 s**
4	**0.244493 s**
8	**0.222464 s**
16	**0.229543 s**
32	**0.229677 s**
64	**0.234422 s**
128	**0.232665 s**

3. For input being 1 000 000 as the size of the array, we evaluate the time taken based on the different number of processes as shown in Table3.
While in sequential logic, when N=500 the time consumed is 2.59000 s in parallel for 8 processes is 0.006483 s. So speedup is 380.88.

Table 3: For N=1 000 000, the number of processes v/s time.

N=1 000 000	
Processes	**Time in s**
2	**2.119033 s**
4	**2.396746 s**
8	**2.313191 s**
16	**2.349319 s**
32	**2.231096 s**
64	**2.273086 s**
128	**2.297322 s**

IV. RESULTS

As input size is Less than equal to 64, Tim sort is performed and it undergoes Insertion Sort as shown in Fig. 4.

Fig.4: Insertion Sort if the array size is less than 64

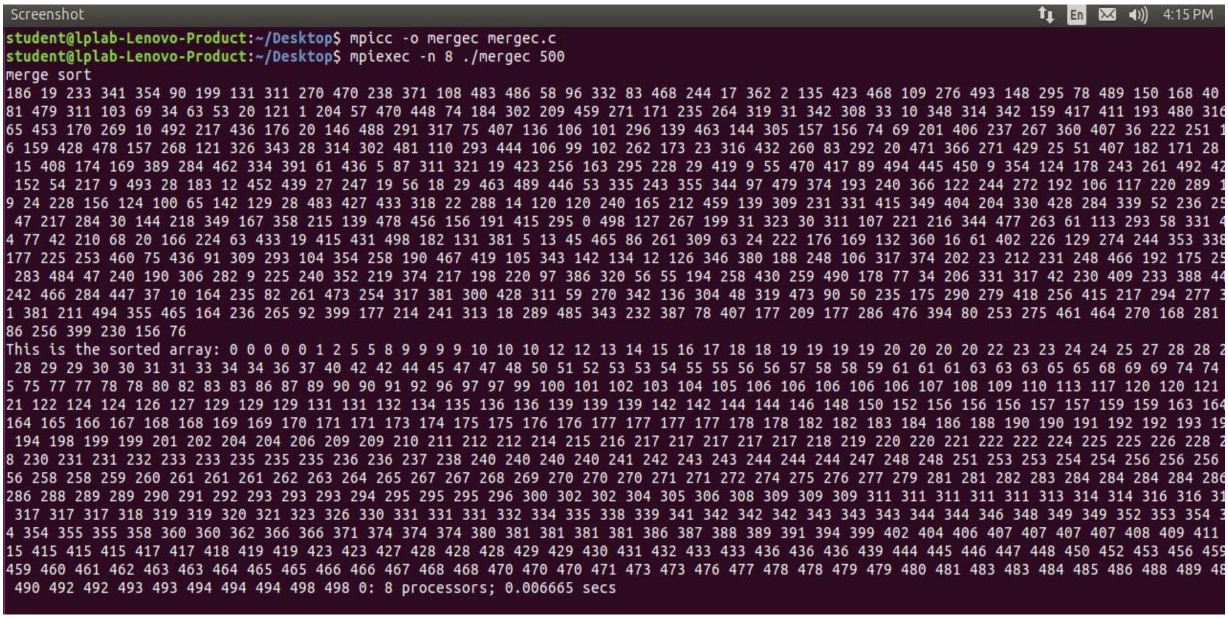

Fig. 5. Merge Sort when the array size is less than 64

If input size is greater than 64, Tim sort is performed and it undergoes Merge Sort as shown in Fig.5.

V. CONCLUSION

Tim sort is a parallel hybrid sorting algorithm that takes in the features of merge sort and insertion sort. Merge sort is optimal on huge data set asymptotically, but on small data set overhead occurs. And for smaller data set insertion sort is best to be chosen. For better performance, if the divide and conquer algorithm is used then for smaller data set best optimal solution is obtained by using insertion sort. So a mixture of merge and insertion sort acts as a good hybrid sorting algorithm thereby allowing Tim sort to have far negligible than O(n log n) comparison because it takes benefit that subarray is may already be sorted.

References

[1] http://en.wikipedia.org/wiki/Timsort[Accessed on::23-Sept-2019]
[2] http://bugs.python.org/file4451/timsort.txt [Accessed On:04-June-2020]
[3] D. E. Knuth. The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching.

Addison Wesley Longman Publish. Co.,
Redwood City, CA, USA, 1998

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein,“Introduction to Algorithms”. MIT Press, Cambridge, MA, 2ndEdition, 2001.

[5] https://media.geeksforgeeks.org/wpcontent/uploads/insertionsort.png[Acccessed On:22-october-2019]

[6] https://en.wikipedia.org/wiki/Merge_sort#/media/File:Merge_sort_alg orithm_diagram.svg [Accessed On:03-October-2019]

[7] http://stromberg.dnsalias.org/strombrg/sort-comparison [Accessed on:23-Sept-2019]

[8] S. Rani, "To Solving Hybrid Appoarch by using Tim Sort Algorithm and Comparision with Quick Sort Algorithm",in MIT International Journal of Computer Science and Information Technology, 2015, pp. 66-70.