Cutshort

by Harshit Yadav

Submission date: 03-Oct-2019 03:05PM (UTC+0800)
Submission ID: 1185168372
File name: check.docx (83.58K)
Word count: 1639
Character count: 8041

***Abstract*— Many sorting algorithms have been developed over the years and the main aim is to reduce the time and space complexity for sorting the worst and average case scenarios. Parallel computing greatly decreases the processing time and increases the processing speed. In this paper we introduce a hybrid algorithm named CutShort algorithm using a parallel processing framework called MPI (Message Passing Interface). We have tested the proposed technique with random samples of large input data. 50% speedup is achieved with parallel processing as compared to the sequential program.**

Keywords—CutShort Algorithm, Message Passing Interface, Parallel processing.

I. INTRODUCTION

Applications of sorting are found almost everywhere. Sorting plays a very important role in almost all the algorithms in Computer Science. There are various sorting algorithms developed each serving its own purpose. Sorting helps in increasing the time complexity for searching, insertion, deletion in many data structures. In this paper we will talk about the CutShort algorithm which works on the bit count operation. We have been learning from school days that any number having more digits is always greater than a number have less no of digits. The main principle of this algorithm is that two numbers can be compared based on the no of digits. This comparison can be applied to elements in an array i.e. it helps in sorting an array of elements.

CutShort Algorithm is a hybrid algorithm which is a combination of the CutShort with Merge sort or Insertion sort or Quick sort. The word "CutShort" is self-explanatory i.e. cutting an input array into smaller pieces of array. These sub arrays are then fed to Merge sort or quick sort or Insertion sort algorithm. Thus the time complexity is reduced by a small amount.

Message Passing Interface (MPI) creates a parallel environment for processing the computations in parallel. In MPI, messages are passed among the various processes created during runtime in order to perform the specific task by each process. This framework work quite well with sorting techniques like merge sort as the sorting application sorts the data locally on processes and later passes the data to its neighbors to process the merged lists.

II. LITERATURE SURVEY

While doing literature survey, we went through various algorithms like Insertion sort, Merge Sort, Quick sort and the time complexities of each one of them. The traditional Insertion Sort algorithm has an average and worst case complexity of O (n^2). When this algorithm is used with CutShort Algorithm the time complexity is reduced to O (nlogn) in average and worst case time complexity.
Further studying the parallel computing framework using MPI, we proposed a model which would combine the CutShort Algorithm with MPI framework which would greatly improve the efficiency and achieve scaleup.

Below sections summarizes the implementation of the proposed model and the experimental comparisons with sequential and parallel approach.

III. METHODOLOGY

In this section we will discuss in detail the working of CutShort algorithm and the Message Passing Framework. We will see in detail the proposed framework of using CutShort Algorithm using MPI and observe how the speedup can be achieved in a parallel environment.

A. CutShort Algorithm

CutShort algorithm works on four steps as mentioned below.
1. Initial Step
2. Range Definition Step
3. Rearranging Step
4. Sub Array sorting Step.

We have discussed each of the steps in a very detailed manner below.

(i) Initial step

In this step, we count the no of bits of a digit in its binary equivalent form. An array of decimal numbers is given as the input. These decimal values are converted into their binary equivalent and their count is stored in an array called Bitband array. With the help of this operation we get to know the no of integers in the input array having the same no of bits.

Consider the input array:

0	1	2	3	4	5	6	7	8	9
22	40	60	52	78	58	18	7	46	82

Fig1: Input Array.

Calculation of the BitCount operation is as shown below:

Table 1: Output of BitCount Operation

Number	Binary Representation	O/P of BitCount operation.
22	10110	5
40	101000	6
60	111100	6
52	110100	6
78	1001110	7
58	111010	6
18	10010	5
7	111	3
46	101110	6
82	1010010	7

In the above example we have one integer with 3 bits i.e. (5), two integers with 5 bits i.e. (22, 18), five integers with 6 bits i.e. (40, 60, 52, 58, 46) and 2 integers with 7 bits i.e. (78, 82). So the BitBand array will be formed as follows:

0	1	2	3
3	5	6	7

Fig2: BitBand Array

(ii) Range Definition Step

In this step, the range refers to the upper and lower boundary of the sub array having only the elements that have the same Bit Count. It is calculated using the follow procedure.
The ranges of these sub arrays can be written as follows: Element 7 will have range from [0, 3), Elements {0, 6} have range from [3, 5), Elements {1, 2, 3, 5, 8} have range from [5, 7), Elements {4,9} have range from [6, 7). The range values calculated by the above method are stored in the BitBand array as shown below:

[0,3)	[3,5)	[5,6)	[6,7)
3	5	6	7

Fig3: Resultant BitBand Array

(iii) Rearranging Step

In this phase, the elements are rearranged according to the no of bit count values. Integers with same bit count are placed together one after the other in a sequential manner in an array.
After this step, the input array would look as given below:

0	1	2	3	4	5	6	7	8	9
7	22	18	40	60	52	58	46	78	82

Fig4: Resultant Input array after Rearranging Step.

With the help of this step, the integers with higher bit count are present on the right side of the array while the integers with less bit count values are present on the left side of the array. In this step the integers with same bit count values are not sorted. After the final step of Sub-array Sorting, the entire input array will be sorted.

(iv) Sub-Array Sorting Step

Now each of these sub arrays can be sorted using any one of the sorting techniques by using the BitBand array that defines the different ranges of bits present in the Input Array. Insertion Sort, Quick Sort or Merge sort can be used for sorting with this BitBand Array.

After this step we get the sorted input array as shown below:

0	1	2	3	4	5	6	7	8	9
7	18	22	40	46	52	58	60	78	82

Fig5: Sorted Input Array

B. *Message Passing Interface*

Message Passing Interface (MPI) is not a programming language but it is a framework that creates an environment for parallel programming by providing libraries in C/C++ Programming. Message Passing Interface (MPI) provides a rich set of abstractions for inter-process communication. MPI supports pair-wise communication (such as using send, receive) and group communication (such as using reduction, barrier). MPI provides a standard interface to the programmers that allow them to write parallel applications that are supported across various platforms. MPI has become the standard communication interface for legacy scientific applications. The primary reason for MPI's success is its wide availability. MPI is available on large scale supercomputers, cloud computing systems and it can also be used for interprocess-communication on a single compute node if other shared memory programming models are not available. Due to the performance reasons, we considered MPI to be the primary communication interface instead of other communication subsystems. Specifically, we have used several MPI routines for our implementation.

C. *Proposed Model (CutShort Algorithm with MPI)*

In the proposed MPI implementation of the cutshort algorithm we parallelize the sequential algorithm by dividing the total input array into sub arrays of equal sizes based upon the number of processors passed as an argument during the runtime , then BitCount function is performed on each element of the sub array in parallel by each processor and the result is stored in separate Bitmap array for each process which is then collected together in the root process and combined together to obtain the final Bitmap array for the given input array . the obtained bitmap array is used for Rearranging step and resultant can be sorted with any sorting algorithm of choice

IV. EXPERIMENT

For the Experimental setup for the proposed model a linux machine was used used having Intel i5-3210M 2.5Ghz processor with 6 GB of 1600Mhz DDR3 RAM and Ubuntu operating system.

Test Case	Quick Sort + CutShort	Insertion Sort + CutShort	Merge Sort + CutShort
1	0.004342	0.035820	0.006833
2	0.004319	0.033692	0.006644
3	0.004367	0.033484	0.006186
4	0.004256	0.028709	0.005831
5	0.004086	0.031815	0.006000
6	0.004119	0.029502	0.005037
7	0.003823	0.026361	0.005370
8	0.003522	0.028389	0.005399
9	0.003773	0.027373	0.005233

Table 1: Time Taken by Different Serial Algorithms (in sec)

For the tests a sample space of ten thousand elements was taken which was divided into three categories for effective analysis of the algorithm in all cases

A.Worst Case [test case 1-3]
All elements of sample space are in range from 2^i to 2^{i+1}

B.Random Values [test case 4-6]
Containing all randomly chosen numbers

C.Best Case [test case 7-9]
All the elements can be equally divided into equal bit range buckets.

Now for Parallel tests the sample space of ten thousand elements was taken which was divided into three categories as mentioned with the use of 4 processors in MPI for the experiment.

Test Case	Quick Sort + CutShort	Insertion Sort + CutShort	Merge Sort + CutShort
1	0.002298	0.027697	0.003462
2	0.002864	0.027697	0.003855
3	0.002310	0.025781	0.003534
4	0.002770	0.022981	0.003526
5	0.002460	0.025772	0.004421
6	0.002301	0.024669	0.003821
7	0.001903	0.024542	0.003662
8	0.002016	0.023974	0.003214
9	0.001900	0.024508	0.003343

Table 2: Time Taken by Different Parallel Algorithms (in sec)

V. Results and Analysis

The following speedup and efficiency were observed when comparing the sequential cutshort algorithm execution compared with its parallel execution

VI. Conclusion

In this paper, we proposed a sorting technique called CutShort Algorithm using MPI. Parallel processing increases the processing speed and can be concluded from the various experiments that we performed.
We achieved a speedup of as compared to running the algorithm sequentially. Further research can be carried out to implement this hybrid algorithm with NVidias CUDA framework to increase the processing speed.

Cutshort

ORIGINALITY REPORT

16%	7%	14%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Ashish Garg, Sudhir Goswami, Versha Garg. "CutShort: A hybrid sorting technique", 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016 Publication	7%
2	arxiv.org Internet Source	4%
3	Chenxu Wang, Tingting Cai, Guang Suo, Yutong Lu, Enqiang Zhou. "DistForest: A Parallel Random Forest Training Framework Based on Supercomputer", 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2018 Publication	2%
4	Submitted to University of Lincoln Student Paper	1%
5	"Encyclopedia of Parallel Computing", Springer	

	Nature, 2011 Publication	1%
6	export.arxiv.org Internet Source	1%
7	epdf.pub Internet Source	1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		