Comparison of CutShort: A Hybrid Sorting Technique using MPI and CUDA

Shraddha Naik   and   Harshit Yadav

Department of Computer Science and Engineering.
Manipal Institute of Higher Education.
Manipal, Karnataka, India, 576104.

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 compare the results of a hybrid algorithm named CutShort algorithm using a parallel processing framework namely MPI (Message Passing Interface) and CUDA (Compute Unified Device Architecture). We tested the proposed technique with random samples of large sample data. 30% speedup is achieved with parallel processing as compared to the sequential program.

Keywords: *CutShort Algorithm, Message Passing Interface, Parallel processing, CUDA.*

1.1 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.

1.2 Literature Survey

While doing literature survey, we went through various algorithms like Quick sort, Merge Sort, Insertion sort, and the time complexities of each one of them. The traditional Insertion Sort algorithm which has an worst case and average case complexity of O (n*n). 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 scale up.

1.3 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 named Bitband array. With the help of this operation we get to know the number of values in the input array having the same no of bits used to represent them.

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, 6), Elements {4,9} have range from [6, 7).The range value is calculated using the method given above and are stored in the BitBand array as shown below:

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

a) 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 2	2 8	1 0	4 0	6 2	5 8	5 6	4 8	7 2	8

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. Merge Sort, Quick Sort or Insertion 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 8	1 2	2 0	4 6	4 2	5 8	5 0	6 8	7 2	8

Fig5: Sorted Input Array

B. MPI (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. MPI provides a large set of abstractions for inter-process communication. It supports group communication (reduction, barrier)and pair-wise communication (send, receive) and . MPI provides a standard interface to the programmers that allow them to write parallel applications that are supported across various platforms. It can also be used on a variety of hardware from a large scale super computer to a single compute node for inter process-communication in a desktop setup. Due to the performance reasons, we considered MPI to be our only communication interface instead of other communication subsystems. Many of several MPI routines are used for our implementation.

C. CUDA (Compute Unified Device Architecture)

CUDA is a parallel computing platform and API Model created by Nvidia. The CUDA platform is designed to work with programming languages such as C, C++ and FORTRAN. This availability makes easy for developers in parallel programming to use GPU to use GPU resources. CUDA program consists of 1 or more phases that are executed on either CPU or a device such as GPU. The phases that exhibit little or no parallelism are implemented on host code. The phase that requires rich amount of parallelism is implemented on device code. Nvidia C compiler separates the two phases during the compilation process. The host code is a straight ANSI C Code and it is further compiled with the host standard C Compilers as an ordinary CPU. The device code is written using ANSI C extended with keywords for labeling data parallelism functions called kernels and their associated data structure. CPU has 1 thread for execution. The device code is compiled by NVCC (Nvidia C Compiler) and executed on GPU device. The kernel typically generates a large n of threads to exploit data parallelism. Due to great performance of CUDA, we implemented this paper using CUDA to improve the efficiency of the algorithm.

D. 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 every value stored at index of the sub array in parallel by each processing element and the BitCount value 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 .

E. Proposed Model (CutShort Algorithm using CUDA)

In the proposed CUDA implementation of the Cutshort algorithm we parallelize the sequential algorithm by sending the input array to the kernel where each thread process the Bitcount function on its respective thread element in parallel and the result is stored in separate bitmap array in global memory which is shared among all the kernel and contain bitmap value obtained from all the parallel execution , the final Bitmap array for the given input array obtained is used in Rearranging step and the result can be sorted with any sorting algorithm of choice.

1.4   **EXPERIMENT**

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

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

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

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.

Table2: Time Taken by Different Parallel Algorithms(in sec)

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

Test Case	Quick Sort + CutShort	Insertion Sort + CutShort	Merge Sort + CutShort
1	0.001706	0.016513	0.003116
2	0.002139	0.016508	0.003144
3	0.001684	0.016697	0.003272
4	0.002232	0.015384	0.003096
5	0.001546	0.018439	0.003192
6	0.001604	0.017452	0.003074
7	0.001108	0.015622	0.002464
8	0.001832	0.014504	0.002258
9	0.001296	0.014912	0.002402

Table3:Time Taken by Different Parallel Algorithms in CUDA (in sec)

1.5   RESULTS AND ANALYSIS

The following speedup and efficiency were observed when comparing the sequential cutshort algorithm execution compared with its parallel execution.
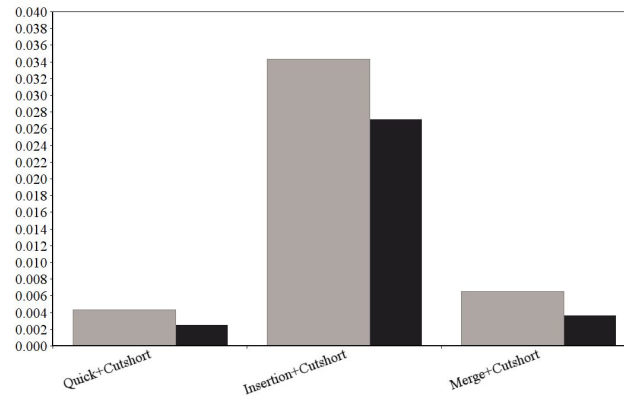
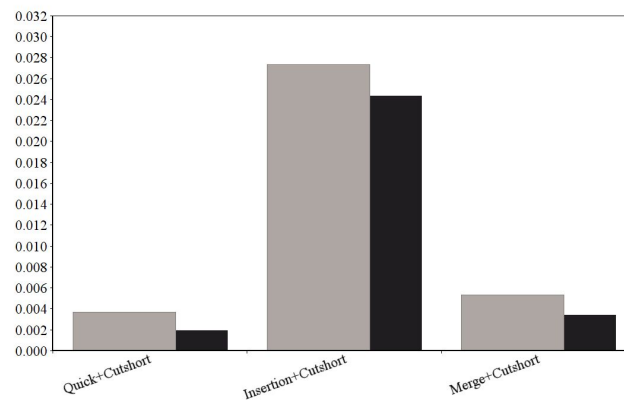Figure 1: Serial & Parallel Execution of Algorithms in Best Case CUDA

Figure 2: Serial & Parallel Execution of Algorithms in Best Case in MPI

1.6   **Conclusion**

In this paper, we proposed a parallel implementation of Cutshort Algorithm. Parallel processing increases the processing speed and can be concluded from the various experiments that we performed.

We achieved a speedup of greater than 30% by implementing parallely in CUDA or MPI as compared to running the algorithm sequentially with the same data set . .

References

[1]   Thomas H. Coreman, Charles E. Leiserson and Ronald L. Rivest, *Introduction to Algorithms,* MIT Press, Third edition, 2009.

[2]   Traditional insertion algorithm – en.wikipedia/wiki/Insertion_sort, three lines implementation and five-lines optimized version by Jon Bentley (1999), Programming Pearls. Addison-Wesley Professional.

[3]   Bender, Michael A; Farach-Colton, Martín; Mosteiro, Miguel (2006), Insertion Sort is O(n log n) SUNYSB; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.3758

[4]   Link: en.wikipedia.org/wiki/Sorting_algorihm

[5]   RCT Lee, SS Tseng, RC Chang and YT Tsai, Introduction to the Design and Analysis of Algorithms, Mc Graw Hill, 2005

[6]   Kazennov, A. (2010). Basic concepts of CUDA technology. Computer Research And Modeling, 2(3), 295-308. doi: 10.20537/2076-7633-2010-2-3-295-308.

[7]   Kirk, D., & Hwu, W. Programming massively parallel processors.

[8]   Using MPI: Portable parallel programming with the message-passing interface. (2000). Computers & Mathematics With Applications, 40(2-3), 419. doi: 10.1016/s0898-1221(00)90207-4