MSD Radix Sort on Chinese Characters Using Pinyin Format

(December 2021)

Vipin Mamidi (001582139) **Satya SriSowmya Chinimilli (002102974)** **Srividya Burra (002985163)**

[1] Northeastern University, Boston, Massachusetts, 02115
[2] College of Engineering

“This work was supported by various other reference materials, reports and citations.”

ABSTRACT

Sorting is an important concept in Programming. It is efficient and faster to find an element in a sorted list than in an unsorted list. There are several sorting algorithms such as counting sorts, string sorts, radix sorts, etc. Imagine having a large data set to sort, counting sorts or the comparison-based sorts gives the worst performance, whereas Radix sort gives linear time. There are two approaches in implementing Radix sort, one is the Most Significant Digit(MSD) sort, which sorts the elements digit by digit from left to right. The other is the Least Significant Digit(LSD) sort, which sorts the elements from right to left. MSD string sorts are attractive because they can get a sorting job done without necessarily examining all of the input characters.

INDEX TERMS: Radix Sort, LSD(Least Significant Digit), MSD(Most Significant Digit), Dual-Pivot Quick Sort(DPQ), Pure Husky Sort, Pinyin order of sorting Chinese dataset.

I. INTRODUCTION

Every binary object is defined as a sequence of bits. In many cases, the order in which we want to sort is identical to the alphabetical order of binary strings.

• Examples: – Sorting regular strings of characters.

• (If by alphabetical order we mean the order defined by the ‘pinyin order’ function, where "Cat" comes before "Coat", because the letters are sorted based on a dictionary format or for example the telephonic directory.)

II. MSD RADIX SORT

MSD Radix sort is a sorting algorithm that has its own interesting characteristics.

• If the radix is R, the first pass of radix sort works as follows:

– Create R buckets.

– In bucket M, store all items whose most significant digit (in R-based representation) is M.

– Reorder the array by concatenating the contents of all buckets.

• In the second pass, we sort each of the buckets separately. – All items in the same bucket have the same most significant digit.

– Thus, we sort each bucket (by creating sub buckets of the bucket) based on the second most significant digit.

• We keep doing passes until we have used all digits.

The string sorting algorithm proposed by Bentley and Sedgwick in 1997 is called Three-Way Radix Quicksort. It works for unbounded alphabets, for which it achieves optimal performance

Theorem 1

The algorithm three-way radix quicksort sorts k strings of total length N in time O(K log K + N). A more precise version of the bounds above (upper as well as lower) is K log K+D, where *D* is the sum of the lengths of the *distinguishing prefixes* of the strings. The distinguishing prefix d_s of a string s in a set S is the shortest prefix of s which is not a prefix of another string in S (or is s itself, if s is a prefix of another string). Clearly, K≤D≤N.

The Three-Way Radix Quicksort of Bentley and Sedgewick is not the first algorithm to achieve this complexity—however, it is a very simple and elegant way of doing it.

Radix sort algorithms fall into two major categories, depending if they process the keys from left to right or backward, from right to left. The forward scanning algorithm is called top-down radix sort or left radix sort. The backward scanning algorithm is called the bottom-up radix sort or the right radix sort.

MSD is a left radix algorithm, which splits the keys into subgroups. The algorithm is applied recursively for each subgroup separately, with the first digits removed from consideration. After the ith step of the algorithm, the input keys will be sorted according to their first I digits.

LSD is a right radix algorithm, which splits the keys into groups according to their last digits, with the last digits removed from consideration after the ith iteration. LSD is non-recursive and must use a stable loop to rearrange the keys. MSD needs only to scan the distinguishing prefixes, while the entire key is scanned in LSD. There is no way to inspect fewer digits in MSD and still be sure that the keys are correctly sorted.

ARL is a new in-place left radix sorting algorithm. It removes the extra space requirement of MSD and uses a different loop than MSD to rearrange the keys. This loop is called the permutation loop in ARL. ARL also uses an adaptive digit-size.

LSD does not partition the input array and therefore it cannot call a different sorting algorithm for small subgroups

III. LITERATURE SURVEY

We have taken 2 technical papers to understand the procedure and functionalities of sorting algorithms that can be implemented and executed with multiple factors.

The gist of our understanding of paper-I i.e., MSD RADIX STRING SORT ON GPU is Efficient sorting is a key requirement for many computer science algorithms. Acceleration of existing techniques as well as developing new sorting approaches is crucial for many real time graphics scenarios, database systems, and numerical simulations to name just a few. It is one of the most fundamental operations to organize and filter the ever-growing massive amounts of data gathered on a daily basis. While optimal sorting models for serial execution on a single processor exist, efficient parallel sorting remains a challenge.

This study proposes a GPU implementation of Radix sort using the Thrust library and Cuda SDK that results in a significant speed up over existing sorting implementations and is by far the most efficient and successfully presented. While thrust performs sorting based on LSD radix sort i.e, Least Significant Digit to benefit from GPU architecture features, it can deal only with fixed length keys. This solution features MSD radix sort, bucket sort that allows for variable length keys and is recursive that makes it difficult to implement on GPU. Also, tried to achieve comparable performance by performing the sorting in several stages which use different parallelization schemes depending on the size and number of buckets. This solution also benefits from shorter alphabets and would be particularly efficient for say, Genomic data.

This report which we made on this presents a hybrid MSD radix sort algorithm for GPU and shows that GPUs can cope with the problem of sorting string data. While MSD Radix sort is less efficient for short/fixed sizes keys, it is suitable for variable-size and long keys. This solution has achieved 60 million strings per second sorting throughput. It is twice as fast as the same algorithm run on a 12-core CPU and 20 times faster than string sorting done with standard sorting routine. It shows that the same parallelization technique is applied to traditional multicore processors too and yields high sorting performance.

For Paper-II, we took MSL: AN EFFICIENT ADAPTIVE IN-PLACE RADIX SORT ALGORITHM, we wrote a paper based on our understanding of this paper which presents an in-place pseudo linear average case radix sorting algorithm.

The proposed algorithm, MSL (Map Shuffle Loop) is a modification of the ARL algorithm. The MSL permutation loop is faster than the ARL counterpart since it searches for the root of the next permutation cycle group by group. The permutation cycle loop maps a key to its target group and shuffles the input array.

The performance of MSL is compared with Java quicksort, as well as MSD and LSD radix sorting algorithms. In-place radix sorting includes ARL and MSL. The run time of MSL is competitive with other radix sorting algorithms, such as LSD. Future work on in- place radix sorting algorithms includes engineered in-place radix sorting. In addition, testing in-place radix sorting algorithms on 64 and 128 bits integers is important.

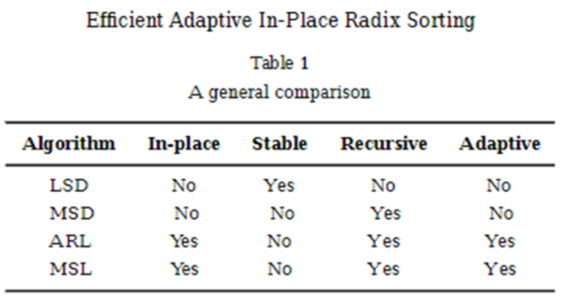
The original Papers had been kept in References, Links to our papers which we wrote our understanding on these published papers :

Paper_1_MSD RADIX STRING SORT ON GPU

Paper_2_MSL: An Efficient Adaptive In-Place …

IV. MATH

MSL uses extra space for groups (buckets). For each group, MSL computes and stores
the boundary or the limit addresses. Each group has a lower limit address, and an upper limit address. The group's sizes are computed before hand and are used to compute group boundary addresses. No extra space is used for the group's sizes, where the upper limit address array is used temporarily instead for this purpose. The formula used for computing the number of groups is 2 digit-size. The digit-size in bits parameter is set prior to any call to MSL. The ARL and the MSL algorithms contribute to radix sorting algorithms, by adding a new category, which we call in-place radix sorting. LSD and MSD are not in-place.



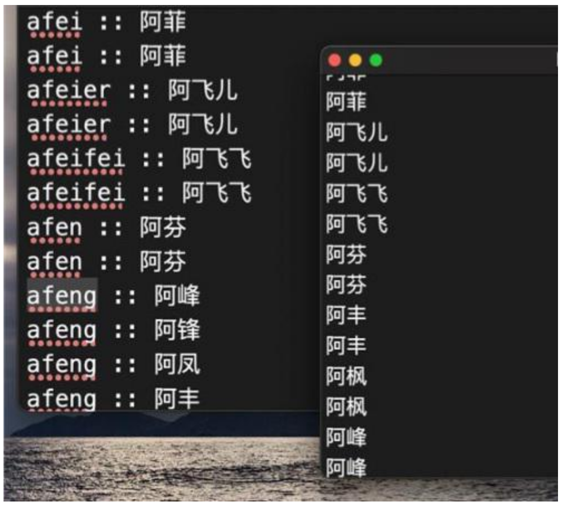
Efficient Adaptive In-Place Radix Sorting

Table 1
A general comparison

Algorithm	In-place	Stable	Recursive	Adaptive
LSD	No	Yes	No	No
MSD	No	No	Yes	No
ARL	Yes	No	Yes	Yes
MSL	Yes	No	Yes	Yes

V. IMPLEMENTATION AND GRAPHS

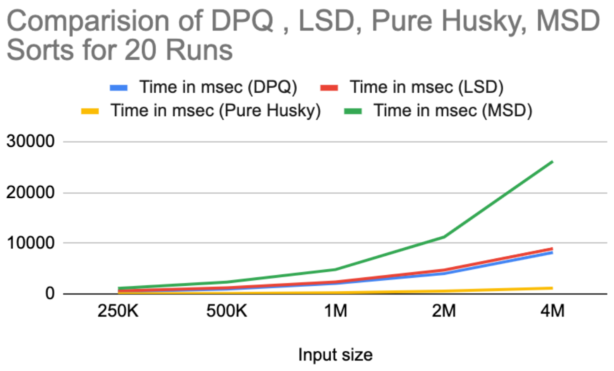
We have implemented the 'pinyin4j' library to extract english letters for the given chinese characters. The class "HanyuPinyinStringarray{}" provides a method "getpinyin()" where we can get the english letters. We pass the input to this class and set the formatting where in this, we used the "With_ToneNumber".
Having the converted English letters, we have passed these letters to the MSD sort method while adding these English letters and corresponding Chinese characters in a multimap data structure.
As we get the sorted english words from the MSD sort method, we use the multimap to print the corresponding chinese words for each english word from the sorted list.
As per our implemented idea of sorting in pinyin order with tone number, we have observed that there are some different chinese characters which have the same pinyin order. Attaching a screenshot for reference. Although the order of the english words after implementing our MSD sort gave a correct result, which is sorted English words.



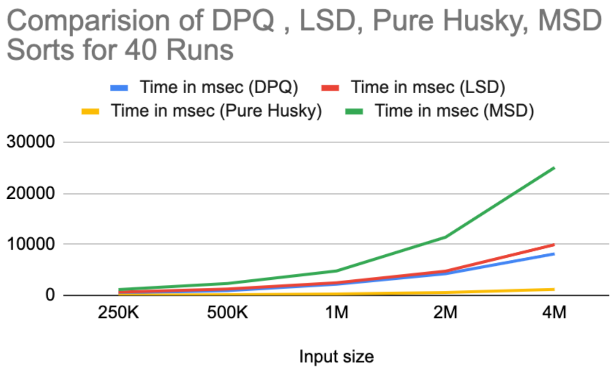
```
afei :: 阿菲
afei :: 阿菲
afeier :: 阿飞儿
afeier :: 阿飞儿
afeifei :: 阿飞飞
afeifei :: 阿飞飞
afen :: 阿芬
afen :: 阿芬
afeng :: 阿峰
afeng :: 阿锋
afeng :: 阿凤
afeng :: 阿丰
```

```
阿菲
阿飞儿
阿飞儿
阿飞飞
阿飞飞
阿芬
阿芬
阿丰
阿丰
阿枫
阿枫
阿峰
阿峰
```

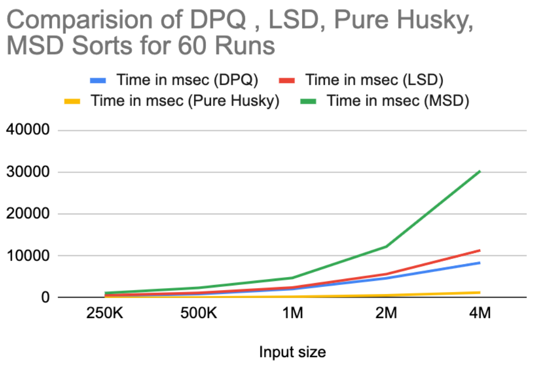
To test our algorithm we have run it with different sizes of arrays varying from 250k, 500k, 1M, 2M upto 4M.Below are the observations made using benchmarking to calculate the time taken by these 4 algorithms - Dual Pivot Quick Sort, LSD Sort, PureHusky Sort and MSD sorting.



Graph plotted for different input array sizes and time taken for respective sorting algorithms for benchmarking of 20 Runs.



Graph plotted above for different input array sizes and time taken for respective sorting algorithms for benchmarking of 40 Runs.



Graph plotted above for different input array sizes and time taken for respective sorting algorithms for benchmarking of 60 Runs.

VI. CONCLUSION

In this paper we have proposed another way to sort Chinese strings that conforms with the proper structure of Chinese words. Our main effort was to maintain the correctness in order set by the Chinese Pinyin order while sorting and to preserve the general complexity of the standard sorting algorithm. We have done the benchmarking of varying sizes of the input array with 250k, 500k, 1M, 2M and 4M datasets. This is to understand how our algorithm works. We have drawn a comparison of our implemented MSD sort with Dual Pivot Quick Sort, LSD sort and Husky Sort. also sorted the algorithm with 1 million, 2 million, 4 million data taken from a chinese dataset.
The output is completely in proper sequence with the pinyin as represented. Thus, this algorithm can be considered as the standard procedure for sorting Chinese strings based on UTF-8.

VII. REFERENCES

Robert Sedgewick and Kevin Wayne , "Algorithms 4th Edition," USA.

http://index-of.es/Varios-2/Algorithms%204th%20Edition.pdf

https://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf

http://vlm1.uta.edu/~athitsos/courses/cse2320_summer2014/lectures/13_radix_sort.pdf

Maus, A.: "ARL: A Faster In-place, Cache Friendly Sorting Algorithm", Norsk Informat- ikkonferranse, NIK '2002, (2002) 85-95.

https://ipsj.ixsq.nii.ac.jp/ej/index.php?action=pages_view_main&active_action=repository_action_common_download&item_id=96611&item_no=1&attribute_id=1&file_no=1&page_id=13&block_id=8

https://link.springer.com/content/pdf/10.1007%2F978-3-540-24687-9_83.pdf