Literature Survey on Radix Sort

1567176 Danlin Lu, 1586011 Wenqi Yang, 1563549 Kunhui Zhang

1. Abstract

This article is a literature survey on radix sort, concluding and summarizing from two technical papers on radix sort. In this article, a brief background of string sorting methods will be introduced at first, followed by basic concepts of the methods. Finally, three more efficient string sorting methods related to radix sort will show up, accounting for the summary of the three papers.

2. Background

Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines, and its algorithm became common for sorting punched cards in 1923.

Computerizing radix sorts were considered impractical because there was no sufficient allocation of buckets of unknown size. However, Harold H. Seward from MIT used a linear scan, which is closely related to counting sort, to solve this problem.

Nowadays, radix sorts are often applied on binary strings and integers, and it has been shown in some benchmarks to be faster than other more general-purpose sorting algorithms.

3. Radix Sort

3.1 Basic Concept

Radix sorts are separated into two types: least significant digit (LSD, right to left) and most significant digit (MSD, left to right). And because of the characteristics of characters: variable lengths and left to right reading style, MSD is commonly used for sorting strings.

The basic concepts for MSD string sort method are partitioning strings and sorting them recursively, while LSD string sort is directly sorting strings based on radix. MSD is more efficient than LSD as it does not have to examine each digit of a string or number. However, it consumes more spaces because of recursion.

To be more efficient, there are three more radix sorts: radix exchange sort in binary machine, Load Balanced Parallel Radix Sort and Partitioned Parallel Radix Sort.

3.2 Radix Exchange Sort

Radix exchange sort is designed for a fixed word length, internally binary machine. The core concept of this method is sorting words containing just the key and an address locating the remainder of the item. The sorting method is checking the key from the highest digit to the lowest digit, similar as MSD radix sort. After the sort, the original data is put in the determined order.

In details, this sorting method with R radix distributes items into R pockets repeatedly. Unless distributing each digit of the key in advance, each of the R pockets must be sufficiently large to accommodate all the items. Each end of the area is treated as the beginning of a separate area, thus only one area of length which equal to that of F items to be sorted is needed. As it is internally binary machine, there are “zeros” pocket and “ones” pocket. Initially, all successive items occupy “zeros” pocket, and when they are working backward, they occupy “ones” pocket. Items are scanned sequentially in the zeros pocket. Encountering "one" is a symbol that “ones” pocket is being scanned. The scanning starts with the last item in the last block and ends when a zero is encountered. Then, the two items are exchanged and being scanned continuously, until all items in the block have been examined. Eventually, all items with bit zero precede all those with bit one in the sorted block.

3.3 Load Balanced Parallel Radix Sort

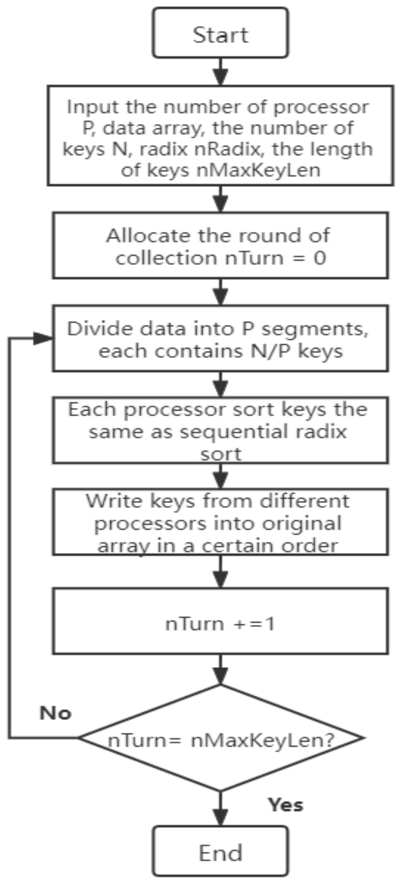
Load Balanced Parallel Radix Sort is a kind of parallel sorting, which is designed to solve the load balancing problem caused by the inconsistent number of keys in each processor. It is based on sequential radix sort, using multiple processors working together to improve efficiency.

Each round of Load Balanced Parallel Radix Sort has two steps. First, we allocate keys into different buckets, and then we collect the data together from different buckets. In allocation step, we divide data into p (the number of processors) segments, and each processors sort the data separately. However, in collection step which is different from sequence radix sort, we need to collect data from all processors and write them into original arrays one by one. Then we just need to repeat the above steps until all the data is sorted.

3.4 Partitioned Parallel Radix Sort

Partitioned Parallel Radix Sort is a kind of parallel radix sort which modifies the Load Balanced Radix Sort to shorten the execution time. The difference is that the Load Balanced Radix Sort requires data redistribution in each round for perfect load balancing, but Partitioned Parallel Radix Sort only needs one in the first round. The remaining work is only computation and data movement within each processor.

To be specific, we assume that the number of processors is P, the total number of keys is N and the number of buckets in each processor is b. The sorting process has two phases, key partitioning and local sort (sequential sort). In key partitioning phase, first, we divide keys into P processors and each processor has N/P keys. Then, we sort the data into b buckets in each processor according to the most significant digit. Finally, we collect keys in different processors according to the value of buckets. In this way we will get b groups keys. The key partitioning phase is similar as the Load Balanced Radix Sort because we need to redistribute data. In local sort phase, we use sequential sort to sort keys in each group. The sorting bases on the other digits except the most significant one. After all sorting is completed, we can output the data in each group one by one and then get the final result.


Start
Input the number of processor P, data array, the number of keys N, radix nRadix, the length of keys nMaxKeyLen
Allocate the round of collection nTurn = 0
Divide data into P segments, each contains N/P keys
Each processor sort keys the same as sequential radix sort
Write keys from different processors into original array in a certain order
nTurn +=1
nTurn= nMaxKeyLen?
No
Yes
End

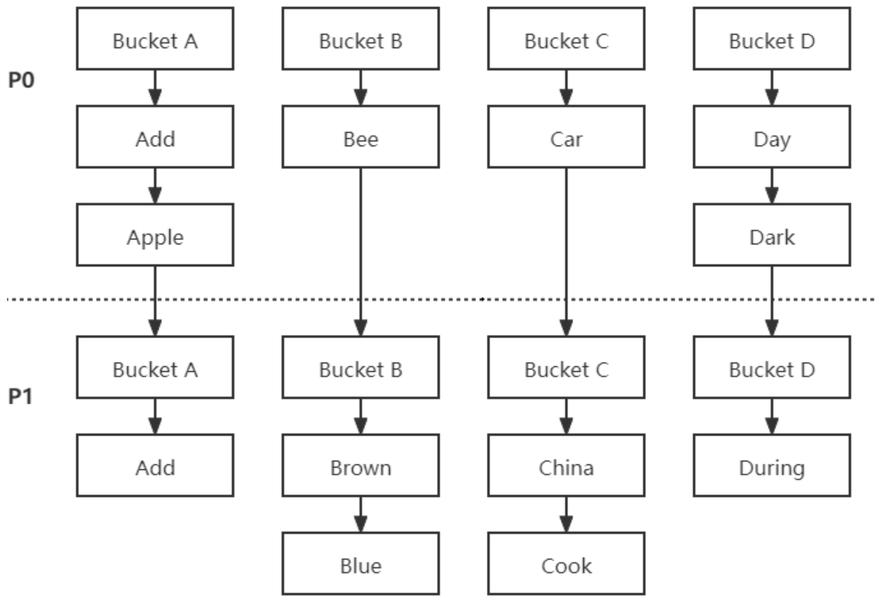


Figure 1 The Algorithm of Load Balanced Parallel Radix Sort

Figure 2 An example of Partitioned Parallel Radix Sort

4. Conclusion

As for radix exchange sorting method, it is useful in situations where space is at a premium. Even where there is ample room in memory, it may prove to be faster than possible alternatives.

The Load Balanced Radix Sort is regarded as one of the fastest internal sorts with parallel processing which requires data redistribution in each round for perfect load balancing. The minor disadvantage of this method is that it takes a lot of time to write all the data into the original array because if we don't do that, we need to traverse the discrete data during the allocation step, which wastes more computational resource.

Theoretically, it seems like that Partitioned Parallel Radix Sort spends fewer computational resources than the Load Balanced Radix Sort. However, although it has a good performance in a multi-machine distributed environment, Partitioned Parallel Radix sort does not do well in a multi-core system because the sequential radix sorting in different buckets waste a lot of computational resources.

5. Reference

1. Paul H. & Harold I. (1959). Radix Exchange – An Internal Sorting Method for Digital Computers. *Journal of the ACM*, 6(2): 156-163.
2. Andrew S. & Yuetsu K. (1998). Load Balanced Parallel Radix Sort. ICS '98: Proceedings of the 12th international conference on Supercomputing, 305-312.
3. Shinjae L. & Minsoo J. & Dongseung K. & Andrew S. (2002). Partitioned Parallel Radix Sort. Journal of Parallel and Distributed Computing, 62(4): 656-668.