Program Structure and Algorithms (INFO 6205)

Fall – 2021 Final Project

Literature Survey on MSD Radix Sort

Keyur Ashokbhai Barot – 001568664

I. Introduction:

Sorting is a fundamental problem in computer science that underlies a vast variety of computational tasks. When the sort keys are strings, it is possible to use any comparison-based sorting algorithm but there are more efficient algorithms specialized for sorting strings. Among the best-known, simplest, and fastest string sorting algorithms is the MSD (Most Significant Digit first) radix sort. In this paper I have tried to summarize the work of two technical papers beginning with [2] and then going on to [3], which shows different methods to sort strings in a more efficient manner.

Radix sort is a non-comparative sorting algorithm. It avoids comparison by creating and distributing elements into buckets according to their radix. For elements with more than one significant digit, this bucketing process is repeated for each digit, while preserving the ordering of the prior step, until all digits have been considered. For this reason, radix sort has also been called bucket sort and digital sort.

Radix sorting algorithms fall into two major categories, depending on whether they process bits left to right, or right to left. After the *kth* step, MSD (Most Significant Digit) sorts the input keys according to their left most *k* digits. Similarly, LSD (Least Significant Digit) processes digits in the opposite direction, but it is non-recursive, unlike MSD.

II. MSL: An Efficient Adaptive In-Place Radix Sort Algorithm:

1. Abstract:

This paper presents an in-place pseudo linear average case radix sorting algorithm. The proposed algorithm, MSL (Map Shuffle Loop) is a modification of the ARL (Adaptive Left Radix) 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.

2. Description:

This method uses the MSL algorithm, whose first steps are presented. Each step executes a loop whose time complexity is indicated. MSL is a modification of the ARL algorithm. Both algorithms use permutation cycles, which is the primary reason for the in-place property of the algorithms. MSL and ARL differ in how they search for and select the root of the next permutation cycle, and in how they compute the radix size. N is the size of input data, while K is the number of groups.

Step 1: Compute groups sizes. O(N)
Step 2: Compute groups start and end addresses. O(K)
Step 3: Permutation cycles (Map shuffle) loop. O(N)
Step 4: Process groups recursively. O(K)

Radix computing in MSL is like Adaptive MSD (AMSD). AMSD radix setting is simple to understand and is modified to work on bits instead of digits. The following modifications are applied to the implementations of both MSL and AMSD: (1) Recompute the radix size continually on recursive calls instead of only once, initially. (2) Use multiple of 4 radix size values and disallow the use of small radix size values, 1 to 3, to improve the running time. The radix sizes used in MSL are 4, 8, and 12, etc. In addition, the implementations of MSL and AMSD use one iteration of quick sort partitioning before calling insertion sort. This initial call inserts the smallest key of each of the two partitions in its correct place. This speeds up insertion sort, which is called frequently. Insertion sort in this case has a redundant if-condition that is removed from its inner loop.

3. Conclusion:

MSL is compared with AMSD, LSD, and Java tuned quicksort. For all four algorithms, the run time improves as the range is decreased. MSL run time is more than twice as fast as Java quicksort, for all runs. MSL run time is better than AMSD. MSL and LSD running time are relatively superior compared to Java quicksort and AMSD. In-place radix sorting includes ARL and MSL. The run time of MSL is competitive with other radix sorting algorithms, such as LSD.

III. Engineering Radix Sort for Strings:

1. Abstract:

This implementation of MSD radix sort is efficient at sorting large collections of strings. These implementations are significantly faster than previous MSD radix sort implementations, and in fact faster than any other string sorting algorithm on several datasets. This also describe a new variant that achieves high space-efficiency at a small additional cost on runtime. Sorting is a fundamental problem in computer science that underlies a vast variety of computational tasks. When the sort keys are strings, it is possible to use any comparison-based sorting algorithm but there are more efficient algorithms specialized for sorting strings

2. Description:

This implementation reduces the number of slow memory accesses through better utilization of the cache memory. In addition, these algorithms reduce the cost of slow memory accesses by better utilization of the out-of-order execution capabilities of modern CPUs.

Algorithmic caching reduces cache misses by copying characters in advance to a place where they can be accessed more efficiently. More precisely, it represents each string not only by a pointer to the beginning but also by four characters stored next to the pointer. Initially, the characters are the first four characters of the string, but later they will be replaced by the next four characters, then by next four and so on. The four characters in the cache are moved with the string pointer and can be accessed efficiently. Only every fourth level of recursion is slower due to refilling the cache.

The idea in the Superalphabet technique is to treat pairs of characters as single characters in a larger alphabet. This effectively halves the number of characters and thus the number of character accesses. Increasing the alphabet size from 2^8 to 2^{16} can also reduce the speed of the algorithm due to the cost of iterating through all buckets. To avoid this, it switches from Superalphabet to normal alphabet when the number of strings drops below 2^{16}.

3. Conclusion:

Radix Sort is a clever and intuitive little sorting algorithm. Radix Sort can also take up more space than other sorting algorithms, since in addition to the array that will be sorted, you need to have a sub-list for each of the possible digits or letters. If you

are sorting pure English words, you will need at least 26 different sub-lists, and if you are sorting alphanumeric words or sentences, you will probably need more than 40 sub-lists in all. Also, it depends on the digits or letters, so for every different type of data, the programmer must rewrite the sorting algorithm, which is quite a big drawback.

IV. References:

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

2. El-Aker F., Al-Badarneh A. (2004) MSL: An Efficient Adaptive In-Place Radix Sort Algorithm. In: Bubak M., van Albada G.D., Sloot P.M.A., Dongarra J. (eds) Computational Science - ICCS 2004. ICCS 2004.

3. Kärkkäinen J., Rantala T. (2008) Engineering Radix Sort for Strings. In: Amir A., Turpin A., Moffat A. (eds) String Processing and Information Retrieval. SPIRE 2008.