'Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set in Distributed Computing Environment'

Avinash Shukla, Anil Kishore Saxena

ABSTRACT

We have proposed a Modified Pure Radix Sort for Large Heterogeneous Data Set. In this research paper we discuss the problems of radix sort, brief study of previous works of radix sort & present new modified pure radix sort algorithm for large heterogeneous data set. We try to optimize all related problems of radix sort through this algorithm. This algorithm works on the Technology of Distributed Computing which is implemented on the principal of divide & conquer method.

1. INTRODUCTION

Sorting is a computational building block of fundamental importance and is the most widely studied algorithmic problem. The importance of sorting has lead to the design of efficient sorting algorithms for a variety of architectures. Many applications rely on the availability of efficient sorting routines as a basis for their own efficiency, while some algorithms can be conveniently phrased in terms of sorting. Database systems make extensive use of sorting operations. The construction of spatial data structures that are essential in computer graphics and geographic information systems is fundamentally a sorting process. Efficient sort routines are also a useful building block in implementing algorithms like sparse matrix multiplication and parallel programming patterns like Map Reduce. It is therefore important to provide efficient sorting routines on practically any programming platform, and with the evolution of new computer architectures there is a need to explore efficient sorting techniques on them. Acceleration of existing techniques as well as developing new sorting approaches is crucial for many real-time graphics scenarios, database systems, and numerical simulations. While optimal sorting models for serial execution on a single processor exists; efficient parallel sorting remains a challenge. Radix sort is classified by Knuth as "sorting by distribution". It is the most efficient sorting method for alphanumeric keys on modern computers provided that the keys are not too long. Floating number sorting is also possible, with same modifications. Radix sort is stable, very fast and is an excellent algorithm on computers having large memory. The idea of radix soft is similar to the idea of hashing algorithms. The final position of the record is computed for each key. If there is already a record(s) with this key, it is placed after them (overflow area). The key is not compared with other keys at all. The approach is generally known as "bucket sorting", "radix sorting," or "digital sorting," because it is based on the digits on the keys.

There are two approaches of radix sorting.

1.1 MSD (most-significant-digit) Radix Sort

Examine the digits in the keys in a left-to-right order, working with the most significant digits first, MSD radix sorts partition the file according to the leading digits of the keys, and then recursively apply the same method to the sub files

1.2 LSD (least-significant-digit) Radix Sort

The second class of radix-sorting methods examine the digits in the keys in a right-to-left order, working with the least significant digits first. Radix sort is work on the radix of elements then the Number of passes depends on the maximum length of elements following are observed.

- For the data set with uniform length, Radix Sort work highly efficiently.
- For data set with unequal length elements, number of passes increases because depending on the maximum length of elements in list, thus increasing Space & Time Complexity.
- In the case of string, strings are sorted but it is corrupted data.

2. REVIEW OF LITERATURE

Nilsson [2] Re-evaluated the method for managing buckets held at leaves & shows better choice of data structures further improves the efficiency, at a small additional cost in memory. For sets of around 30,000,000 strings, the improved burst sort is nearly twice as fast as the previous best sorting algorithm. **Jon l. Bentley** [3] suggested a detailed implementation combining the most effective improvements to Quick sort, along with a discussion of how to implement it in assembly language. It is wide applicability as an internal sorting method which requires minimal memory. **Arne Anderson** [4] had presented and evaluated several optimized and implemented techniques for string sorting. Forward radix sort has a good worst-case behavior. Experimental results indicate that radix sorting is considerably faster (often more than twice as fast) than comparison-based sorting. It is possible to implement a radix sort with good worst-case running time without sacrificing average-case performance. The implementations are competitive with the best previously published string sorting programs. **Naila**

Rahman [5] Discuss the problem of large applications data set which were too massive to fit completely inside the computer's internal memory. The resulted input/output communication between fast internal memory and slower external memory was a major performance bottleneck. [5] Also discussed the distribution and merging techniques for using the disks independently. These are useful techniques for batched EM problems involving matrices (such as matrix multiplication and transposition), geometric data (such as finding intersections and constructing convex hulls), and graphs (such as list ranking, connected components, topological sorting, and shortest paths) were proposed. In the online domain, canonical EM applications include dictionary lookup and range searching. They also re-examined some of the EM problems in slightly different settings, such as when the data items are moving, or when the data items are variable-length (e.g., text strings), or when the allocated amount of internal memory can change dynamically. **Rajeev Raman** [6] illustrated the importance of reducing misses in the standard implementation of least-significant bit first in (LSB) radix sort, these techniques simultaneously reduce cache and TLB misses for LSB radix sort, all the techniques proposed yield algorithms whose implementations of LSB Radix sort & comparison-based sorting algorithms. **Danial** [7] explained the Communication and Cache Conscious Radix sort Algorithm (C3-Radix sort). C3-Radix sort uses the distributed shared memory parallel programming Models. Exploiting the memory hierarchy locality and reduce the amount of communication for distributed Memory computers. C3-Radix sort implements & analyses on the SGI Origin 2000 NUMA Multiprocessor & provides results for up to 16 processors and 64M 32bit keys. The results show that for small data sets compared to the number of processors, the MPI implementation is the faster while for large data sets, the shared memory implementation is faster. **Shin-Jae Lee** [8] solved the load imbalance problem present in parallel radix sort. Redistributing the keys in each round of radix, each processor has exactly the same number of keys, thereby reducing the overall sorting time. Load balanced radix sort is currently the fastest internal sorting method for distributed-memory based multiprocessors. However, as the computation time is balanced, the communication time becomes the bottleneck of the overall sorting performance. The proposed algorithm preprocesses the key by redistribution to eliminate the communication time. Once the keys are localized to each processor, the sorting is confined within processor, eliminating the need for global redistribution of keys & enables well balanced communication and computation across processors. Experimental results with various key distributions indicate significant improvements over balanced radix sort. **Jimenez- Gonzalez** [9] introduced a new algorithm called Sequential Counting Split Radix sort (SCS-Radix sort). The three important features of the SCS-Radix are the dynamic detection of data skew, the exploitation of the memory hierarchy and the execution time stability when sorting data sets with different characteristics. They claim the algorithm to be 1:2 to 45 times faster compare to Radix sort or quick sort. **Navarro & Josep** [10] focused on the improvement of data locality. CC-Radix improved the data locality by dynamically partitioning the data set into subsets that fit in cache level L2. Once in that cache level, each subset is sorted with Radix sort. The proposed algorithm is about 2 and1:4 times faster than Quick sort and Explicit Block Transfer Radix sort. **Ranjan Sinha** [11] suggested that the Algorithms for sorting large data sets can be made more efficient with careful use of memory hierarchies and reduction in the number of costly memory accesses. Burst sort dynamically builds a small tree that is used to rapidly allocate each string to a bucket. Sinha & Zobel introduced new variants of algorithm: SR-burst sort, DR-burst sort, and DRL-burst sort. These algorithms a-priori construct a tree from random samples. Experimental results with sets of over 30 million strings show that the new variants reduce, by up to 37percent cache misses than the original burst sort, and simultaneously reducing instruction counts by up to 24 percent. **Jian- Jun Han** [12] proposed two contention-aware scheduling algorithms viz. OIHSA (Optimal Insertion Hybrid Scheduling Algorithm) and BBSA (Bandwidth Based Scheduling Algorithm). Both the algorithms start from the inherent characteristic of the edge scheduling problem, and select route paths with relatively low network workload to transfer communication data by modified routing algorithm. OISHA optimizes the start time of communication data transferred on links. BBSA exploits bandwidth of network links optimally. Moreover, the proposed algorithms adapt not only to homogeneous systems but also heterogeneous systems. **Sinha, R. and Zobel** [13 & 14] examined that the Burst sort is a cache-oriented sorting technique using dynamic tree to efficiently divide large sets of string keys into related subsets small enough to sort in cache. In original burst sort, string keys sharing a common prefix were managed via a bucket of pointers represented as a list or array. C-burst sort copies the unexamined tail of each key to the bucket and discards the original key to improve data locality. Results indicate that C-burst sort is typically twice as fast as original burst sort and four to five times faster than multi-key quick sort. CP-burst sort uses more memory, but provides stable sorting. **Nadathur Satish** [15] proposed the high-performance parallel radix sort and merge sort routines for many-core GPUs, taking advantage of the full programmability offered by CUDA. Radix sort is the fastest GPU sort and merge sort is the fastest comparison-based sort reported in the literature. For optimal performance, the algorithm exploited the substantial fine-grained parallelism and decomposes the computation into independent tasks. Exploiting the high-speed on chip shared memory provided by NVIDIA's GPU architecture and efficient data-parallel primitives, particularly parallel scan, the algorithms targeted the GPUs. **N. Ramprasad and Pallav Kumar Baruah** [16] suggested an optimization for the parallel radix sort algorithm, reducing the time complexity of the algorithm and ensuring balanced load on all processor. [16] Implemented it on the "Cell processor", the first implementation of the Cell Broadband Engine Architecture (CBEA). It is a heterogeneous multi-core processor system. 102400000 elements were sorted in 0.49 seconds at a rate of 207 Million/sec. **Shibdas Bandyopadhyay and Sartaj Sahni** [17] developed a new radix sort algorithm, GRS, for GPUs that reads and writes records from/to global memory only once. The existing SDK radix sort algorithm does this twice. Experiments indicate that GRS is 21% faster than SDK sort while sorting 100M numbers and is faster by between 34% and 55% when sorting 40M records with 1 to 9 32-bit fields. **Daniel Jiménez-González, Juan J. Navarro, Josep-L. Larrba-Pey** [18] proposed Parallel in-memory 64-bit sorting, an important problem in Database Management Systems and other applications such as Internet Search Engines and Data Mining Tools. [9] The algorithm is termed Parallel Counting Split Radix sort (PCS-Radix sort). The parallel stages of the algorithm increases the data locality, balance the load between processors caused by data skew and reduces significantly the amount of data communicated. The local stages of PCS-Radix sort are performed only on the bits of the key that have not been sorted during the parallel stages of the algorithm. PCS-Radix sort adapts to any parallel computer by changing three simple algorithmic parameters. [9] Implemented the algorithm on a Cray T3E-900 and the results shows that it is more than 2 times faster than the previous fastest 64-bit parallel sorting algorithm. PCS-Radix sort achieves a speed up of more than 23 in 32 processors in relation to the fastest sequential algorithm at our hands. **Daniel Cederman and Philippas Tsigas** [19] showed at GPU-Quick sort, an efficient Quick sort algorithm suitable for the highly parallel multi-core graphics processors. Quick sort had previously been considered an inefficient sorting solution for graphics processors, but GPU-Quick sort often performs better than the fastest known sorting implementations for graphics processors, such as radix and bitonic sort. Quick sort can thus be seen as a viable alternative for sorting large quantities of data on graphics processors.

1. COMPARISON OF VARIOUS SORTING ALGORITHMS [1]

The following table compares the sorting algorithms according to the complexity, method used by them like exchange, insertion, selection, merge and also discuss their advantages and disadvantages. n represents the number of element to be sorted.

TABLE 1.1: COMPARISON OF COMPARISON BASED SORT

Name	Average Case Time Complexity	Worst Case Time complexity	Method	Advantage/Disadvantage
Bubble Sort	$O(n^2)$	$O(n^2)$	Exchange	1. Straightforward, simple and Stable. 2. Slow & difficult on large data set.
Insertion Sort	$O(n^2)$	$O(n^2)$	Insertion	1. Efficient for small list and Save memory 2. Slow for large data set.
Selection Sort	$O(n^2)$	$O(n^2)$	Selection	1. Improvement over Bubble sort 2. Unstable & very slow for large Data set.
Heap Sort	O(n log n)	O(n log n)	Selection	1. More efficient version of Selection sort. It does not require recursion & extra buffer. 2. Slower than Quick and Merge Sorts.
Merge Sort	O(n log n)	O(n log n)	Merge	1. A fast recursive sorting suitable for very large data set. 2. It requires large memory space.
In place-merge Sort	O(n log n)	O(n log n)	Merge	1.Very low memory required 2. Unstable & slow.

Shell Sort	O(n log n)	O($n\log_2 n$)	Insertion	1. Efficient for large data set, relatively small memory required. 2. It is not stable & has more constants.
Quick Sort	O(n log n)	O(n^2)	Partition	1. Fastest, efficient & required less memory space. 2. Partition can lead to unbalanced.

3.1 COMPARISON OF NON COMPARISON BASED SORTING ALGORITHMS [1]

The following table describes sorting algorithm which are non Comparison sort. Complexities below are in terms of n, the number of item to be sorted, and k the size of each key and s is the chunk size use by implementation. Assume that the key size is large enough, that all entries have unique key values.

3.1 TABLE 1.2: COMPARISON OF NON - COMPARISON SORT [1]

Name	**Average Case**	**Worst Case**	**n<<2K**	**Advantage/disadvantage**
Bucket Sort	O(n.k)	O(n_2.k)	No	1. Stable & fast. 2. Used in special cases when the key can be used to Calculate the address of Buckets.
Counting Sort	O(n+2k)	O(n+2k)	Yes	1. Stable, used for repeated Value & often used as a subroutine in radix sort. 2. Valid for integer only.
Radix Sort	O(n.ks)	O(n.ks)	No	1. Stable, straight forward 2. Applicable to data set with multiple fields.
MSD Radix Sort	O(n.ks)	O(n.ks)	No	1. Highly efficient for sorting large data sets. 2. Bad worst-case performance due to data fragmentation.
LSD Radix Sort	O(n.ks)	O(n.ks.2s)	No	3. Stable & fast sorting method.

2. PROPOSED MODIFIED RADIX SORT

It is observed that no single method is optimal to all available data sets with varying complexity of size, number of fields, length etc. Thus attempt is made to select a set of data set & optimize the implementation by modifying the basic algorithm. Above these problems of Sorting algorithm are optimized by proposed algorithm. The algorithm is dependent on the distributed Computing Environment. Its implementation is proposed on many core machines. Given heterogeneous list is divided into two main process one is numeric and other is string. These two process work simultaneously. Suppose p1, p2 are the two main process. Each process has a unique processor. Process p1 is further distributed in different sub list according to equal length of elements in a list. These lists are sorted simultaneously on the logic of even & odd logic. Passes are transferred alternatively on the digits. After sorting these lists combined all this & again sort this main list. In the case of p2, make a pattern. Using the unique pattern, get the selected strings. Among these strings, same string provides same numeric values. Now proposed algorithm applies on these numeric values for sorting the given strings.

3. RESULTS AND DISCUSSIONS

Now proposed MRS algorithm runs on two different machines & has observed the results. Off course results have shown clear picture that MRS Sort is best sort for heterogeneous data set on both the machines always. After MRS Sort GPU Quick Sort is the best option. Both Sorting techniques are complete by themselves, but there are slight differences between these two sorting methods given below. This algorithm runs on two different machines, the results are as follow in the form of graph.

First this algorithm runs on Intel Pentium P6200,Intel HD Graphics,2GB DDR3 RAM,500 GB HDD Operating system :- Windows 7. The results (Graph Representation) of this machine are as follow.

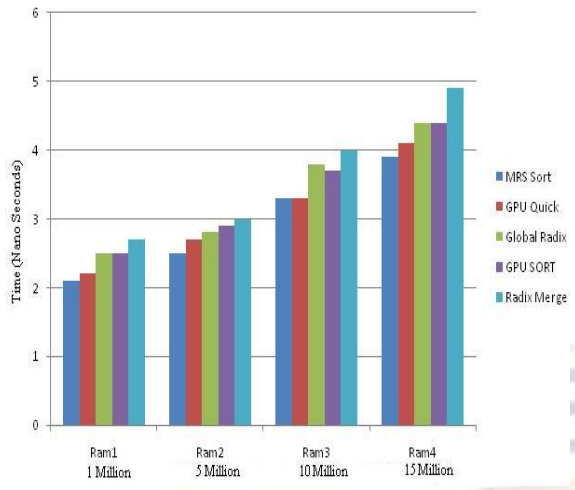

Fig. 1 Comparisons Results of MRS Sort on machine 1

Here four groups are presents whose name like Ram1, Ram2, Ram3, Ram4. Each group keeping separate heterogeneous data set. Ram1 represent 1 millions of heterogeneous data set like Ram1 other groups keeping 5 millions, 10 millions, 15 millions & 20 millions heterogeneous data set respectively. All these groups are shown on X- axis on the graph & y-axis shown on taking time (Nano Seconds) for each group.

1. Second time this algorithm runs on Intel Xeon Server Board, Intel HD Graphics, 5GB DDR3 RAM, 500 GB HDD, Operating system : - windows server 2008 R2. The results (Graph Representation) of this machine are as follow.

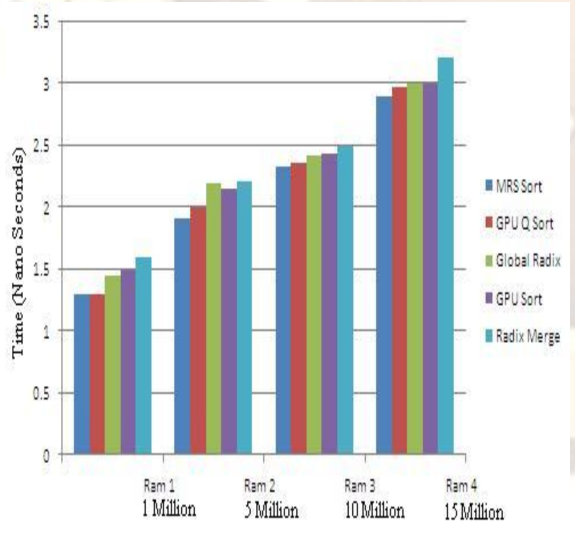

Fig. 2 Comparisons Results of MRS Sort on machine 2

Here four groups are presents whose name like Ram1, Ram2, Ram3, Ram4. Each group keeping separate heterogeneous data set. Ram1 represent 1 millions of heterogeneous data set like Ram1 other groups keeping 5 millions, 10 millions, 15 millions & 20 millions heterogeneous data set respectively. All these groups are shown on X- axis on the graph & y-axis shown on taking time (Nano Seconds) for each group. The results are clearly shown some condition MRS Sort & GPU Quick Sort are same results & some condition MRS Sort just up on GPU Quick Sort algorithms.

4. CONCLUSION

Now, clearly seen that given algorithm can do much better job over existing sorting algorithms. Both time & space complexities are optimized with this algorithm. The various algorithm prepared so far for sorting of large heterogeneous data set are discussed. It can be seen that none of the algorithm is optimized universally for all types of data set. Thus approach to develop optimized algorithm for affliction data set are being discussed and proposed. A new Algorithm proposed optimized the time & space complexity for heterogeneous data set comprising of both alphanumeric, string & available in different format. The results had shown an improvement of 10:20% in computational complexity compound with MRS sort & GPU Quick sort.

RRFERENCES

1. PhD. Thesis by Aditya Dev Mishra under the supervision of Dr. Deepak Garg CSED CSE Department Thapar University Patiala-147004 June 2009 on the Topic" Selection of Best Sorting Algorithm for a particular problem.
2. Nilsson, S. 1996," Radix sorting & searching", Ph.D. thesis, Department of Computer Science, Lund University, Lund, Sweden.
3. Jon L. Bentley, Robert Sedgwick, "Fast algorithms for sorting and searching strings", Proceedings of the eighth annual ACM-SIAM symposium on discrete algorithms, p.360-369, January 05-07, 1997, New Orleans, Louisiana, United States.
4. Arne Anderson, Stefan Nilsson, "Implementing radix sort", Journal of Experimental Algorithmic (JEA), 3, p.7-es, 1998.
5. Naila Rahman, Rajeev Raman, "Analyzing cache effects in distribution sorting", Journal of Experimental Algorithmic (JEA), 5, p.14-es, 2000.
6. Naila Rahman, Rajeev Raman, "Adapting Radix Sort to the Memory Hierarchy", Journal of Experimental Algorithmic (JEA), 6, p.7-es, 2001.
7. Danial, Navarro, Guinovart & Larriba pay," Sorting on the SGI Origin 2000: Comparing MPI & Shared memory Implementations", 19th IEEE Conference of the Chilean Computer Science Society (1999) current version available on 6th August 2002.
8. Shin-Jae Lee, Minsoo Jeon, Dongseung Kim & Andrew Sohn, "Partitioned Parallel sort 1", IDEAL Journal of Parallel & Distributed

Computing (2002) jpdc.

9. Danial, Jimenez- Gonzalez, J. J. Navarro, Josep [2002]", the Effect of Local Sort on Parallel Sorting Algorithms", 10th IEEE Euromicro Workshop on Parallel Distributed & Network Based Processing (Euromicro-PDP' (02) 2002.
10. Daniel, Navarro & Josep, "CC- Radix: a Cache Conscious Sorting Based on Radix Sort", 11th IEEE Conference on Parallel, Distributed & Network-Based Processing (Euro-PDP'03) 2003.
11. Ranjan Sinha, Justin Zobel," Cache-conscious sorting of large sets of strings with Dynamic Tries", Journal of Experimental Algorithmic (JEA), 9, 2004.
12. Sinha, R. and Zobel, J. 2004. "Using random sampling to build approximate tries for efficient string sorting", Springer- Verlag International Workshop on Efficient and Experimental Algorithms, vol. 3059. New York. 529-544.
13. Sinha, R. 2006. "Using compact tries for cache-efficient sorting of integers". Springer-Verlag International Workshop on Efficient and Experimental Algorithms, C. C. vol. 3059, New York. 513-528.
14. Jian- Jun Han & Duo- Qiang wang, "Edge Scheduling Algorithm in parallel & Distributed Systems", IEEE Conference on parallel Processing (ICPP'06) 2006.
15. Nadathur Satish "Designing Efficient Sorting Algorithm for many-core GPUs", 23rd IEEE International Parallel and Distributed Processing Symposium, May 2009.
16. N. Ramprasad and Pallav Kumar Baruah.2007." Radix Sort on the Cell Broadband Engine" ,In Int.l Conf. High Perf. Comuting (HiPC) – Posters, 2007.
17. Shibdas Bandyopadhyay and Sartaj Sahni. 4-Feb 2011," GPU Radix Sort For Multifield Records", IEEE Explore High Performance Computing(Hipc),2010 International Confrence on19-22 Dec. 2010 ,1-10, Dona Paula, 11824284,Dept. of CSE, University of Florida,Gainesville, FL 32611.
18. Daniel Jiménez-González, Navarro, josep L. Larrba – Pey. 2001."**Fast Parallel in-memory 64-bit sorting**" Proceeding ICS '01 Proceedings of the 15th international conference on Supercomputing ACM New York, NY, USA ©2001.

19. Daniel Cederman and Philippas Tsigas.2008,"On sorting and load balancing on GPUs", Newsletter ACM SIGARCH Computer Architecture News archive Volume 36 Issue 5, December 2008 ACM New York, NY, USA.

ABOUT THE AUTHORS

[1]**Avinash Shukla** was born in Jabalpur on 14th Oct.1979. He has done Msc.in Information Technology from MCNUJC Bhopal in 2007. He is presently doing PhD. in Computer Science & Engineering in from CMJ University, Shilong (Meghalaya), India.

[2]**Dr A.K Saxena, ME,** PhD (Engg.) from ABV- IIITM. He is the Member Board of Studies in Electronics & CSE, RGTU (2005-2008).Electrical Engg., Jiwaji University(1999-2001) and have the Administrative experience as Nominee for BE and MCA courses at various institutes, Assistant coordinator(PET).He is the member of FIETE, MIE, MISTE Societies and have the Industrial Experience at SICO, Indian Railway Construction Co. Ltd. He had presented 10 technical papers in various national and international conferences. His research area includes Multimedia Technology and Digital Watermarking; Microprocessor based system development and instrumentation.