The Detailed Experimental Analysis of Bucket Sort

Neetu Faujdar
Department of CSE
Amity University, Noida
India
neetu.faujdar@gmail.com

Shipra Saraswat
Department of CSE
Amity University, Noida
India
sshipra1510@gmail.com

***Abstract*—The bucket sort is a non-comparison sorting algorithm in which elements are scattered over the buckets. We have concluded, based on state-of-art that most of the researchers have been using the insertion sort within buckets. The other sorting technique is also used in many papers over the buckets. From the state-of-art of bucket sort, we have analyzed that insertion sort is preferable in case of low volume of data to be sorted. In this work, authors have used the merge, count and insertion sort separately over the buckets and the results are compared with each other. The sorting benchmark has been used to test the algorithms. For testing the algorithms, sorting benchmark has been used. We have defined the threshold (τ) defined the threshold for saving the time and space of the algorithms. Results indicate that, count sort comes out to be more efficient within the buckets for every type of dataset.**

***Index Terms*—Merge sort; Count Sort; Insertion Sort; Bucket Sort.**

I. Introduction

The most fascinating computational problem in computer science is sorting. There are two categories of sorting algorithms known as comparison and non-comparison based sorting algorithms. This paper consists the both categories of sorting. Bucket sort is a most popular integer sorting algorithm [1]. The bucket sort uses the some other sorting technique over the buckets to sort the elements. The question arises that which sorting approach should be used in the bucket. The state of art tells us that its depend on the nature of the data. In this paper, we have used the merge, count and insertion sort within the bucket.

Merge sort is a stable divide and conquer technique [2] is based on an array of *n* objects. The array is divided into two halves and applies each half separately [3]. Finally,the sorted sub halves is merged into one sorted array. If the elements are low in volume, then using the insertion sort.

To sort the bridge hands insertion sort is used. The two regions are identified at each iteration. 1) Sorted region and 2) Unsorted region.From the unsorted region, elements are picked and then insert it in sorted region [4]. Every iteration, there is an increase of elements one by one. Similarly the procedure is repeated for the remaining unsorted regions. Astrachan [5] shows in his experiment that bubble sort is 5 times more slower in comparison to insertion sort and 40% slower rate than selection sort which illustrates, that insertion sort is the fastest algorithm among the three.

Count sort is a non-comparison sorting algorithm [6] based on the linear amount of time complexity achieves by the count sort. It is based on the range of key element from 1 to *k* where *k* is the some other integer [7]. The drawback of count sort is that if the range of key element increases, then memory occupied by the algorithm will also increase[8-10].

In this paper, bucket sort is considered for research concern. The paper consists the detailed experimental analysis of bucket sort. The merge, insertion and count sort is used as a local sort for buckets. Following are the key points of the authors of this paper:
1. The main focus of this work is on bucket sort..
2. Merge, insertion and count sort is used separately as a local sort for buckets and the results are compared with each other.
3. We have defined the threshold (τ) in order to save the time and space of the algorithms.
4. Sorting benchmark is used to test the algorithms.
5 The input data *N* is varied from 1000 to 5000000.

The value of 'τ' depends on the range and number of buckets. It is given by the equation described below:

$$\tau = R/B \tag{1}$$

In the above equation *R* is the range of the element that can be easily identified or it is obtained from the prior knowledge of the problem. In general the value R is calculated by the expression given below.

$$R = MAX(A) - MIN(A) \tag{2}$$

Here *A* represents the set of elements that provides the input to the bucket sort and *B* is the number of buckets that are used for sorting.

Rest paper is presented as follows and based on bucket sort is listed in the section 2. Sorting benchmark which is used to test the algorithms listed in the section 3. Implementation details of algorithms used within the bucket is summarized in section 4. The final section 5 listed the conclusion and future work with few remarks.

II. Related Work

Sorting is a very demanding field in computer science. In this section, we briefly survey related work of bucket sort. Panu Horsmalahti compared the radix and bucket sort. He has measured the time usage and memory consumption for different kind of input data. The both theoretical and experimental analysis has been done in the paper. The has been shown that bucket sort was faster than the radix sort, but in some cases more memory is occupied by the bucket sort.

Apostolos Burnetas *et al* presented the specific bucket sort algorithm. The main advantage of the specific bucket sort is the achievement of linear average time. The linked list data structure is not used to sort the elements. The results have shown that the improved computational efficiency is obtained. The data is considered from 1000 to 3000 uniformly distributed positive integers. The developed approach was achieved the efficiency, faster than quick and standard bucket sort [11].

Burak takmaz *et al* determined the some improvement using the some different sorting technique. The author has mixed the bucket and shell sort. The suggested approach worked with greater performance in comparison to the traditional bucket sort [12].

Zhongxiao Zhao *et al* proposed the innovative method by using hash function based on the probability density function. In this method n record is allocated to n number of buckets uniformly based on the value of the key. In this way, the proposed approach achieves the $O(n)$ time under any situation [13].

III. Sorting Benchmark

We have tested the bucket with count sort, bucket with merge sort and bucket with insertion sort on six types of test cases (Uniform, Sorted, Zero, Bucket, Gaussian, and Staggered) [14-22]. The input data is varied from 1000 to 500000 and the size of bucket is kept constant which is 10 for every data. The threshold (τ) is calculated by using the equation 1.

1. Uniform test case: represents test case values chosen between 0 to 2^{32}.
2. Gaussian test case: depicts the distribution of data by taking the average values taken from the uniform distribution.
3. Zero test case: producing the constant value.
4. Bucket test case: works on the basis of $p \in N$, the N shows the input size value which splits the p blocks in the first n/p^2 elements randomly in $[0, 2^{31}/p\text{-}1]$, and the next element is n/p^2 in $[2^{31}/p, 2^{31}/p - 1]$, and so on.
5. Staggered test case: works on $p \in N$, size of N is divide into p blocks in such a manner where block indexing is $\mathrm{i} \leq p/2$ for all its n/p elements setting randomly to $[(2i\text{-}1)2^{31}/p, (2i)(2^{31}/p - 1)]$.
6. Sorted test case: uses the sorted uniform distributed values for further processing.

IV. Algorithms Used

By using sorting benchmark technique, bucket sort algorithm has been tested.Three sorting algorithms have been used as a local sort within buckets which are merge, count and insertion sort. Total six types of test cases have been used by sorting benchmark. The input data N is altered from 1000 to 5000000. The number of buckets has been fixed for every data size which is 10. The threshold is calculated using equation (1). The execution time of bucket with merge sort is summarized in Table I.

Table II, listed the execution time of bucket with count sort in seconds. Here, we have evaluated the six types of test cases during execution time on the basis of sorting benchmark. The zero test case has achieved the lesser execution time among others. It is because we are using one unique value in this test case.

Execution time of bucket with insertion sort is listed in Table III. The zero test case is achieved less execution time in comparison to others. It is because, zero test case have only one constant value.

The Fig 1 to 6 is represented by using the values of Table I, II and III. In the entire Figure 1 to 6:
- M stand for bucket with merge sort.
- C stand for bucket with count sort.
- I stand for bucket with insertion sort.

In Fig 1 to 6, the size of input data is presenting in X-axis while execution time in seconds showing in Y-axis. Key element range of count sort is taken from 0 to 65535.

Fig 1 illustrates,the comparison of execution time based on uniform test cases. The figure infers that inside the bucket insertion sort is preferable only when volume of data is less. The count sort comes out to be best even if we raise the volume of data. It is because, the count sort is based on the range of key element which is used from 0 to 65535. But if the range of key element increases then count sort will be worst in comparison to other sorting algorithms.

Fig 2 illustrates, the execution time comparison of bucket sort using bucket test case. In this test case count sort with in bucket comes out to be more efficient in comparison to merge and insertion sort in every input data. It is because the count sort is based on the range of key elements.

Fig 3 illustrates, the execution time comparison of bucket sort based on Gaussian testcase in which count sort with case bucket is realized to be more efficient in comparison to other algorithms. This provides reliable results because on the basis of wide range of key elements.

Fig 4 shows, the execution time comparison of bucket sort using sorted test case. The figure infers that bucket with insertion sort is more efficient in comparison to bucket with merge sort. The result of bucket with insertion sort comes to nearly equal to bucket with count sort. It is because the best case of insertion sort occurs when data is sorted or nearly sorted.

Fig 5 shows, the execution time comparison of bucket sort using staggered test case. In this test case bucket with count sort is more efficient in comparison to other at every input data. It is because the count sort is based on the range of key elements.

Fig 6 shows, the execution time comparison of bucket sort using zero test case. In this test case bucket with count sort is more efficient in comparison to other at every input data. It is because the count sort is based on the range of key elements.

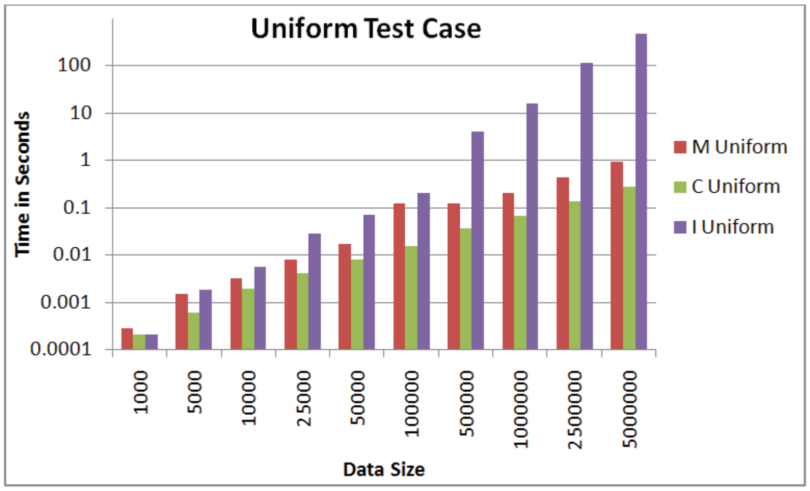

Fig. 1. Execution time comparison of uniform test case

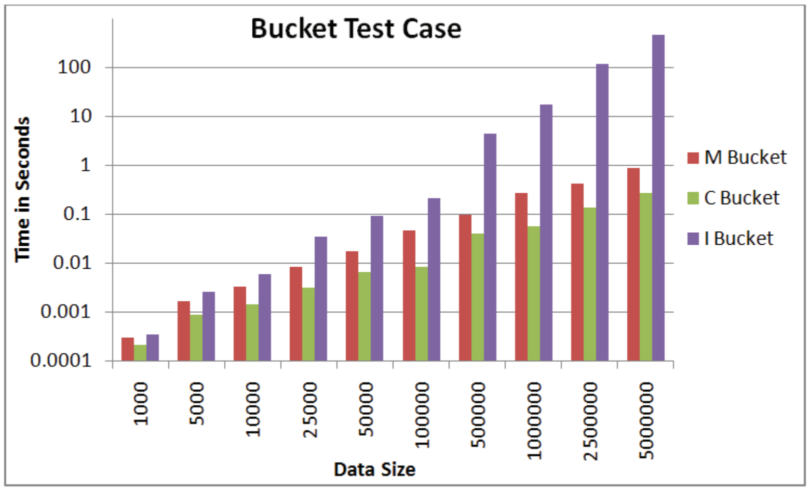

Fig. 2. Execution time comparison of bucket test case

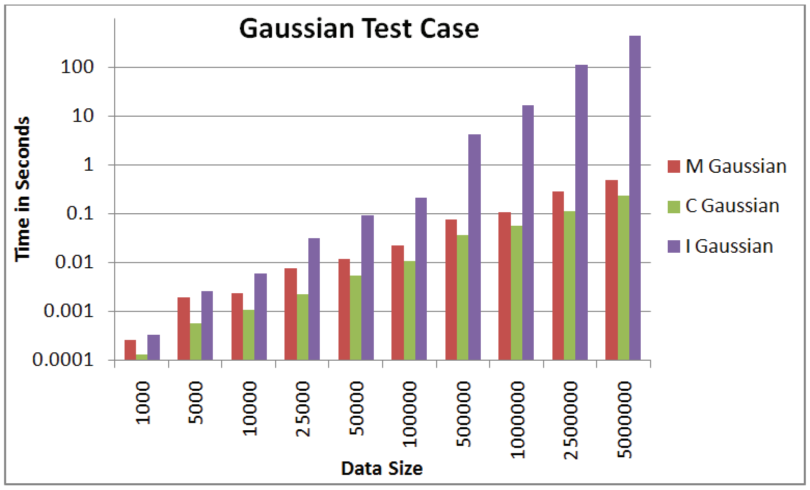

Fig. 3. Execution time comparison of gaussian test case

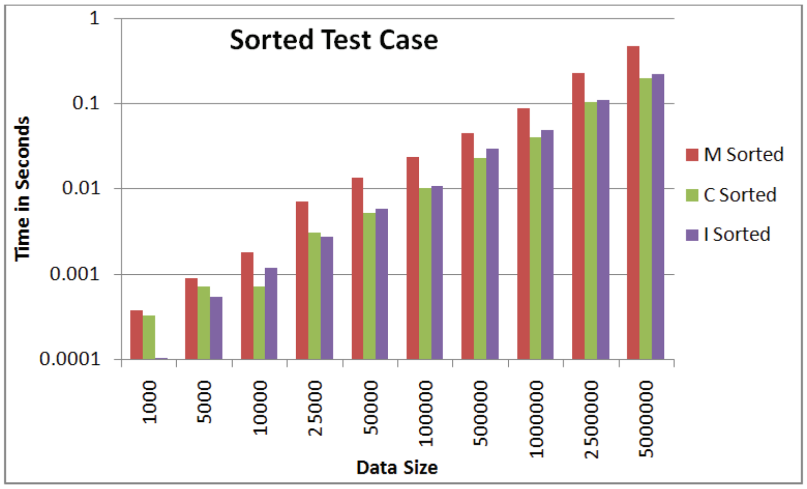

Fig. 4. Execution time comparison on the basis of sorted test case

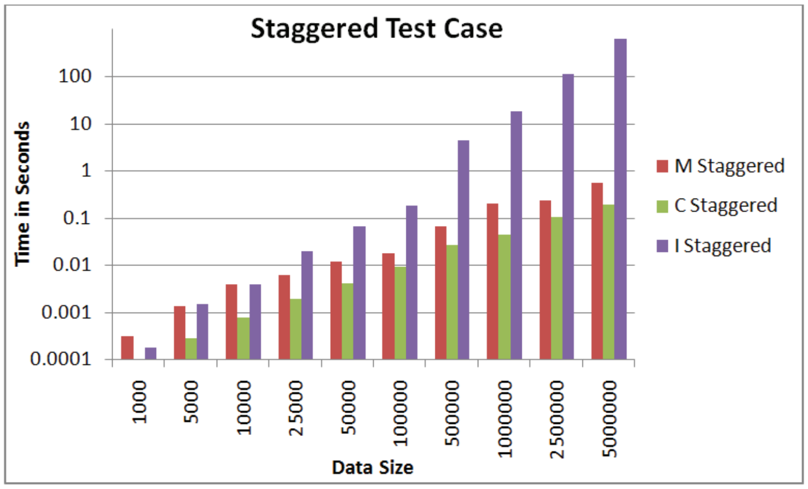

Fig. 5. Execution time comparison on the basis of staggered test case

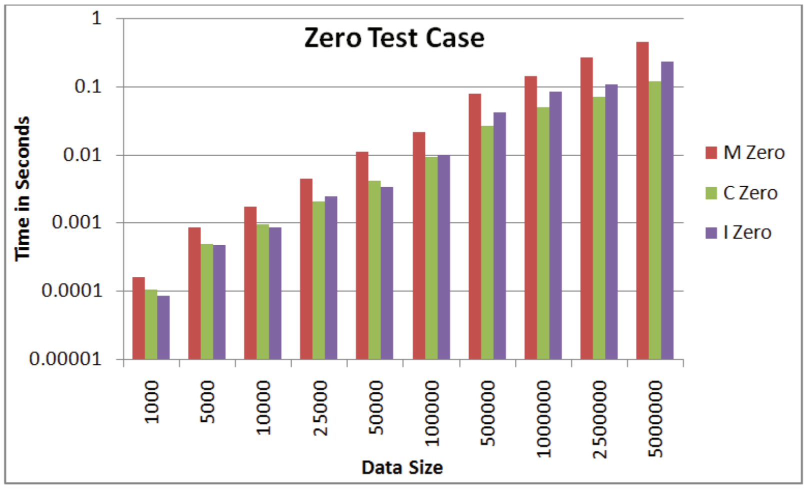

Fig. 6. Time comparison during execution time bases on zero test case

TABLE I
EXECUTION TIME OF BUCKET WITH MERGE SORT IN SECOND

Buckets	Data Size	Threshold	Uniform	Bucket	Gaussian	Sorted	Staggered	Zero
10	1000	100	0.000288	0.000288	0.000255	0.000369	0.000299	0.000157
10	5000	500	0.001561	0.001616	0.001871	0.000893	0.001337	0.000844
10	10000	1000	0.003187	0.003214	0.002345	0.00177	0.003847	0.001706
10	25000	2500	0.008301	0.008167	0.007453	0.007135	0.005971	0.004431
10	50000	5000	0.017326	0.017363	0.011407	0.013258	0.011804	0.010924
10	100000	10000	0.12633	0.045763	0.022332	0.023236	0.017315	0.021636
10	500000	50000	0.12633	0.096887	0.074469	0.044236	0.066083	0.078691
10	1000000	100000	0.202372	0.267329	0.106931	0.088435	0.195731	0.143183
10	2500000	250000	0.428226	0.411191	0.273599	0.229846	0.235101	0.263882
10	5000000	500000	0.921813	0.873912	0.476948	0.473334	0.540467	0.442345

TABLE II
EXECUTION TIME OF BUCKET WITH COUNT SORT

Buckets	Data Size	Threshold	Uniform	Bucket	Gaussian	Sorted	Staggered	Zero
10	1000	100	0.00021	0.000209	0.000126	0.000328	0.000104	0.000105
10	5000	500	0.000628	0.000865	0.000547	0.000717	0.000291	0.00049
10	10000	1000	0.001992	0.001421	0.001055	0.000717	0.000793	0.000954
10	25000	2500	0.004164	0.003123	0.002197	0.003028	0.001979	0.002011
10	50000	5000	0.00803	0.006395	0.005282	0.005227	0.00409	0.004096
10	100000	10000	0.015658	0.008056	0.010737	0.010221	0.00938	0.009118
10	500000	50000	0.037869	0.038936	0.036503	0.022833	0.026704	0.026677
10	1000000	100000	0.066718	0.05518	0.054219	0.040114	0.045151	0.049566
10	2500000	250000	0.139848	0.132886	0.111055	0.103042	0.107318	0.070051
10	5000000	500000	0.277671	0.262354	0.233062	0.198703	0.190298	0.120295

TABLE III
EXECUTION TIME OF BUCKET WITH INSERTION SORT

Buckets	Data Size	Threshold	Uniform	Bucket	Gaussian	Sorted	Staggered	Zero
10	1000	100	0.00021	0.000346	0.000321	0.000103	0.000174	0.000086
10	5000	500	0.00186	0.002523	0.0025	0.000539	0.001449	0.000472
10	10000	1000	0.00578	0.005869	0.005821	0.001168	0.003897	0.000861
10	25000	2500	0.02834	0.033548	0.031541	0.002695	0.019257	0.002471
10	50000	5000	0.07264	0.090566	0.09051	0.00584	0.064391	0.003382
10	100000	10000	0.20351	0.213359	0.20964	0.010866	0.180903	0.009867
10	500000	50000	4.03731	4.369265	4.166221	0.02965	4.350074	0.041963
10	1000000	100000	16.1925	17.43392	16.2184	0.049203	18.39243	0.084605
10	2500000	250000	111.118	113.458	110.663	0.108898	115.5556	0.107356
10	5000000	500000	460.372	469.6966	465.6367	0.221633	633.1936	0.231269

V. CONCLUSION

In this paper, a detailed analysis of bucket sort has been done. The insertion, count and merge sort algorithms have been used within the buckets. For testing the algorithms, sorting benchmark has been used. Three algorithms which are bucket with insertion, bucket with count and bucket with merge sort have been implemented and compared to each other.

The threshold (τ) is defined for saving the time as well as space of the algorithms. Results indicate that, count sort comes out to be more efficient within the buckets for every type of dataset with respect to the range of key elements. The range used in this work is from 0 to 65535. But if the range of key element increases then count sort will be worst in comparison to other sorting algorithms in both aspects (space, time).

Based on Window 7, operating system of 64 bit, core i5 processor of Intel algorithms are tested which are implemented in C-language. Borland C++ 5.02 compiler is used for program designing and verified after running on 2.2 GHz clock speed.

In the future work, we can further classify other sorting algorithm like quick sort based on the number of elements in bucket which will not only make the working faster of the bucket sort but also will reduce the time.

Acknowledgment

Authors are very thankful to the research environment provided by Amity University, Noida, India. All experimental results are verified in the research lab of this organization.

References

[1] Horsmalahti, Panu, “Comparison of Bucket Sort and RADIX Sort,” arXiv preprint arXiv:1206.3511, June 2012.

[2] Radenski, Atanas, “Shared memory, message passing,” and hybrid merge sorts for standalone and clustered SMPs, International Conference on parallel and Distributed processing Techniques and Applications CSREA press, pp. 367-373, 2011.

[3] D. Abhyankar, “A fast merge sort,” Int. Journal of Engineering Research and Applications, vol. 4, pp. 131-134, April 2014.

[4] Kumar, Akash, et al, “Merge Sort Algorithm,” International Journal of Research,” vol. 1, pp. 16-21, 2014.

[5] Owen Astrachan, “Bubble Sort: An Archaeological Algorithmic Analysis,” SIGCSE, 2003.

[6] Neetu Faujdar, Satya Prakash Ghrera, “Performance Evaluation of Parallel Count Sort using GPU Computing with CUDA,” Indian Journal of Science and Technology, vol. 9, DOI: 10.17485/ijst/2016/v9i15/80080, May 2016.

[7] Bajpai, Keshav, and Ashish Kots, & Michael, G. “Implementing and Analyzing an Efficient Version of Counting Sort (E-Counting Sort),” International Journal of Computer Applications, vol. 98, pp. 1-2, January 2014.

[8] Faujdar, Neetu, and Satya Prakash Ghrera, “Analysis and Testing of Sorting Algorithms on a Standard Dataset,” IEEE Fifth International Conference on Communication Systems and Network Technologies (CSNT), pp. 962-967, April 2015.

[9] Neetu Faujdar, Satya Prakash Ghrera, “Performance Evaluation of Merge and Quick Sort using GPU Computing with CUDA,” International Journal of Applied Engineering Research (IJAER), vol. 10, pp. 39315-39319, 2015.

[10] Neetu Faujdar, Satya Prakash Ghrera, “A detailed experimental analysis of library sort algorithm,” Annual IEEE India Conference (INDICON), pp. 1-6, December 2015.

[11] Burnetas, Apostolos, Daniel Solow, and Rishi Agarwal, “An analysis and implementation of an efficient in-place bucket sort,” Acta Informatica, vol. 34, pp. 687-700, September 1997.

[12] Takmaz, Burak, and Murat Akin, “A new approach to bucket sort,” Proceedings of the 7th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, World Scientific and Engineering Academy and Society (WSEAS), pp. 184-186, February 2008.

[13] Zhao, Zhongxiao, and Chen Min, “An Innovative Bucket Sorting Algorithm Based on Probability Distribution,” WRI World Congress on IEEE Computer Science and Information Engineering, vol. 7, pp. 846-850, March 2009.

[14] Cederman, Daniel, and PhilippasTsigas, “Gpu-quicksort: A practical quicksort algorithm for graphics processors,” Journal of Experimental Algorithmics (JEA), vol.14, pp. 1-22, December 2009.

[15] Leischner, Nikolaj, VitalyOsipov, and Peter Sanders, “GPU sample sort,” IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1-10, April 2010.

[16] Matsumoto, Makoto, and Takuji Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator,” ACM Transactions on Modeling and Computer Simulation (TOMACS), vol. 1, pp. 3-30, January 1998.

[17] Neetu Faujdar, Satya Prakash Ghrera, “Modified Levels of Parallel Odd-Even Transposition Sorting Network (OETSN) with GPU Computing using CUDA,” Pertanika J. Sci. & Technol, vol. 24, pp. 331-350, July 2016.

[18] Arturo Garcia, Jose Omar Alvizo Flores, Ulises Olivares Pinto, Felix Ramos, “Fast Data Parallel Radix Sort Implementation in DirectX 11 Compute Shader to Accelerate Ray Tracing Algorithms,” EURASIA GRAPHICS: International Conference on Computer Graphics, Animation and Gaming Technologies, pp. 27-36, 2014.

[19] Peters, Hagen, Ole Schulz-Hildebrandt, and Norbert Luttenberger, “Parallel external sorting for CUDA-enabled GPUs with load balancing and low transfer overhead,” IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1-8, April 2010.

[20] Svenningsson, Josef David, et al, “Counting and occurrence sort for GPUs using an embedded language,” Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing, Boston, MA, USA, ACM, pp. 37-46, September 2013.

[21] Jaumin Ajdari, Bujar Raufi, Xhemal Zenuni, and Florije Ismaili, “A version of parallel odd-even sorting algorithm implemented in cuda paradigm,” International Journal of Computer Science Issues (IJCSI), vol. 12, May 2015.

[22] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, L. Rauchwerger, “A framework for adaptive algorithm selection in stapl,” Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, vol. 12, May 2015.