Performance Analysis of Parallel Sorting Algorithms using GPU Computing

Neetu Faujdar
Dept of CSE
Jaypee University of Information Technology,
Waknaghat Solan, India

SP Ghrera
Dept of CSE
Jaypee University of Information Technology,
Waknaghat Solan, India

ABSTRACT
Sorting is a well interrogating issue in computer science. Many authors have invented numerous sorting algorithms on CPU (Central Processing Unit). In today's life sorting on the CPU is not so efficient. To get the efficient sorting parallelization should be done. There are many ways of parallelization of sorting but at the present time GPU (Graphics Processing Unit) computing is the most preferable way to parallelize the sorting algorithms. Many authors have implemented the some sorting algorithms using GPU computing with CUDA. This paper mentioned the roadmap of research direction of a GPU based sorting algorithms and the various research aspects to work on GPU based sorting algorithms. These research directions include the various sorting algorithms which are parallel (Merge, Quick, Bitonic, Odd-Even, Count, Radix etc.) sort algorithms using GPU computing with CUDA (Compute Unified Device Architecture). In this paper, we have tested and compared the parallel and sequential (Merge, Quick, Count and Odd-Even sort) using dataset. The testing of parallel algorithms is done using GPU computing with CUDA. The speedup is also measured of various parallel sorting algorithms. The results have depicted that, the count sort is the most efficient sort due to based on the key value. Future research will refine the performance of sorting algorithms in GPU architecture.

Keywords
GPU, CUDA, Parallel Sorting Algorithms.

1. INTRODUCTION
GPU stands for Graphics Processing Unit. In the 1999-2000 computer scientist started using the GPU to extend the range of scientific domain. The term GPU was popularized in 1999 by NVIDIA. The world first GPU was a Geforce 256. To do the GPU programming, we require the use of graphics APIs such as OpenGL and WebGL. In 2002 James Fung developed OpenVIDIA. It is used for parallel GPU computer vision. The projects of OpenVIDIA implement computer observation algorithms run on graphics hardware such as OpenGL, Cg and CUDA-C. In November 2006, NVIDIA launched CUDA (Compute Unified Device Architecture). It is an API (Application Programming Interface) that allows coding the algorithms for execution on Geforce GPUs using C as a high level programming language. CUDA can use with other languages. Nowadays, several GPGPU (General Purpose computing on GPUs) languages, such as OpenCL and NVIDIA CUDA are proposed for designers to use GPUs with extended C programming language, instead of graphics API. The modern NVIDIA GPUs are precisely programmable in C using CUDA. The parallel computing with CUDA organizes concepts of Grid, Block and Thread. The threads are classified on the structure of grids, block and threads. The threads are executed using SIMT (Single Instruction and Multiple-Threads) style. In this paper, the roadmap of research direction about GPU based sorted algorithm is suggested. The paper also listed the analysis of GPU parallel sorting algorithms over sequential. The contribution of the paper is as follows.

1. The paper contains the roadmap of research direction about GPU based sorting algorithms.
2. Some parallel sorting using GPU computing is related with the sequential sorting.
3. The speedup achieved by parallel sorting over sequential is also measured in this paper.

The paper demonstrates as follows. In section 2, we have discussed the motivation of the paper. Literature review based on parallel sorting using GPU computing is listed under the section 3. The section 4 shows the hardware configuration of the system used to execute the algorithms. The performance comparison of parallel and sequential Sorting Algorithms is described in section 5. The measurement of speedup is illustrated in section 6. The conclusion and future work is listed under the section 7.

2. MOTIVATION
Nowadays GPU is in big demand in parallel computing. Numerous researchers are working on the GPU. The programmability of the GPU is rising in the world. The rising GPU has enabled the threshold. The following point makes the user to work on the GPU rather than the CPU.

- The speed of CPU is less as compared to GPU.
- GPUs are comically strong.
- GPUs have a troublesome departure path.
- The GPU programming model emerging.
- GPU is densely parallel.
- It has hundreds of cores.
- It has thousands of threads.
- It is cheap
- It is highly available.

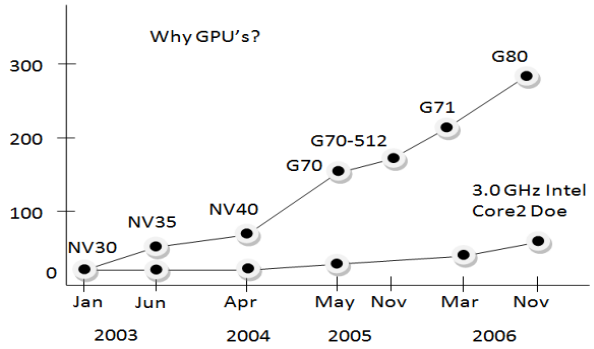

Fig: Rising of GPU

In the Fig. 1 NV stands for NVIDIA GPU and shows that GPUs are highly rising rather than CPU.

3. LITERATURE REVIEW BASED ON PARALLEL SORTING USING GPU

Greb *et al* presented the parallel sorting based on stream processing architecture in the year 2006. The proposed sorting is based on bitonic sort who is adaptive. The optimal time complexity of proposed approach achieved O (n log n)/p). The proposed algorithm is faster than sequential sorting. The proposed algorithm is designed on modern GPU, so the name GPU-ABiSort [1].

Inoue *et al* proposed the AA-sort which is parallel sorting algorithm. AA-sort stands for Aligned-Access Sort. AA-sort proposed for shared memory multiprocessors. The sequential version of the AA-sort is more beneficial for IBM's optimized sequential sorting using SIMD instructions [2].

Sintorn *et al* presented the fast algorithm to sort huge data using modern GPU. The implementation of the algorithm is fast due to the GPU. The proposed algorithm performed better than bitonic sort algorithms for the input list with more than 512k elements. The suggested approach is 6-14 times quicker than the single CPU quick sort of 1-8M elements [3].

Cederman *et al* presented the GPU Quick Sort. The proposed algorithm is extremely capable and suitable for parallel multi-core graphics processors. GPU quick sort performance represents the better performance than the fastest known GPU based sorting algorithms such as radix and bitonic sort [4].

Rozen *et al* presented the adoption of the bucket sort algorithm. The proposed algorithm is entirely run on the GPU. The proposed algorithm is implemented on GPU using OpenGL API [5]

Baraglia *et al* showed that how the graphics processor used as a coprocessor to speed up the algorithm and CPU also allowed doing the some other task. The proposed algorithm is used to memory efficient data access pattern to maintain the minimumnumber of access to the memory of the chip. The implementation results show the improvement in the GPU based sorting in order to CPU based sorting [6].

Leischner *et al* presented the GPU sample sort algorithm. Thesample merge sort is the efficient comparison based sorting algorithm for distributed memory architecture. Previously the sample sort algorithm was unknown for the GPU [7].

Kukunas *et al* presented the GPU merge sort. In today's life high data throughput and computational power are increasing. The GPGPU architecture is created by NVIDIA. The GPU merge sort is highly efficient in comparison to a sequential version [8].

Oat *et al* presented the technique for sorting data into spatial bins using GPU. The proposed technique takes the unsorted data as input and scatters the points in sorted order into the buckets. The author proposed method is used to implement a form of bucket sort using GPU [9].

Huang *et al* proposed the empirical optimization technique. The empirical optimization technique is also important for sorting routines using GPU. The radix sort generated the highly productive code for NVIDIA GPU with a variety of architecture specification. The paper outcome showed that the empirical optimization technique is quite successful. The resulting code was more efficient than radix sort [10].

Ye *et al* presented GPU warp sort to carry out a comparison based parallel sort on the GPU. The warp sort is nothing but contain the bitonic sort followed by merge sort. The proposed algorithm achieved the high staging by depicting the sorting task on the GPU. The experimental results of GPU- Warpsort work well on various kinds of input distribution [11].

Peters *et al* presented the Batcher's bitonic sorting network using CUDA hardware with GPUs. The arbitrary numbers has been taken as input and assigned compare-exchange operation to threads using adapted bitonic sort. The proposed algorithm has greatly increased the performance of implementation [12].

Peters *et al* presented the merge-based external sorting algorithm using CUDA sanction GPUs. The production influence of memory transfer is reduced using GPU. The better utilization of the GPU and load balancing is achieved. The performance of the algorithm is demonstrated by extended testing. The two main problems occur when using external sorting on GPUs [13].

Satish *et al* reported the comparison and non-comparison based sorting algorithms on CPUs and GPUs. The author has extended the work to the Intel Many Integrated Core (MIC) architecture. The radix sort evaluated on Knights Ferry and obtained the performance gain of 2.2X and 1.7 X. The production of the GPU radix sort improves nearly 1.6X over previous outcomes [14].

Helluy presented the portable OpenCL implementation of the radix sort. The algorithm was tested on several GPUs or CPUs in order to access the good performance. The implementation was also applied to the Particle-In-Cell (PIC) sorting. The application of the PIC is plasma physics simulations. The radix sort algorithm contains the following steps: 1) Histogramming. 2) Scanning. 3) Reordering [15].

Krueger *et al* presented a technique, differential updates which are used to permit rapid modifications. The lead storage is allowed to the database to maintain data storage for accommodating the modifying queries. The author also presented the parallel dictionary slice merge algorithm and also GPU parallel merge algorithm that achieves 40% more throughput in comparison to CPU [16].

Mišić *et al* represented an effort of sorting algorithms to analyze and implement in the graphics processing unit. Three sorting algorithms evaluated on the CUDA architecture. The evaluated algorithms are quick, merge and radix sort. CUDA platform used the NVIDIA GPU to execute applications [17].

Peters *et al* presented the novel optimal sorting algorithm which is similar to the adaptive bitonic sort. The popular parallel merge based sorting algorithm is the adaptive bitonic

sort. It uses the tree like data structure to achieve the optimal complexity called a bitonic tree. The author presented the execution of the hybrid algorithm for GPUs based on bitonic sort [18].

Jan *et al* presented examines three extensively used parallel sorting algorithms. The algorithms are Odd-Even sort, rank sort and bitonic sort. The comparative analysis is performed in terms of sorting rate, sorting time and speedup on CPU and different GPU architectures. The author achieved the high speed-up of NVIDIA quadro 6000 GPU for min-max butterfly network reaching much lower sorting for high data [19].

Munavalli developed a novel sorting algorithm on the GPU. Author focused on the vital problem. Author presented an efficient sorting algorithm which is Fine Sample Sort (FSS). The proposed algorithm extends and outperforms the sample sort algorithm. The results have shown that FSS outperforms sample sort by at least by 26% and on average 37% of data size ranging from 40 million and above for various input distributions [20].

Thouti *et al* presented the comparative performance analysis of various sorting algorithms. The algorithms are bitonic and parallel radix sort. Author implemented both the algorithms in OpenCL and compared with the quick sort algorithm. The author used the Intel Core2Due CPU 2.67 GHz and NVIDIA Quadro FX 3800 as GPU for the implementation [21].

Zurek *et al* described the implementation results for a few diverse parallel sorting algorithms using GPU cards and multi-core processors. The author presented the hybrid algorithm and executed on both platforms CPU and GPU. The comparison of many core and multi-core is lacking. The threads are grouped in blocks and the blocks are grouped in grids [22].

Panwar *et al* used the GPU architecture for solving the sorting problem. The highly parallel computing nature of GPU architecture is utilized for sorting purposes. The author considered the input array in the form of 2D matrix which is used for sorting. The modified version of merge sort is applied in that matrix. This work performed much efficient sorting algorithm with reduced complexity [23].

Garcial *et al* presented the fast data parallel implementation of radix sort using the Direct Compute software development kit (SDK). Author also discussed the optimization strategies in detail that are used to increase the performance of radix sort. The paper share the insights should be used in GPGPU (General Purpose Graphics Processing Unit). Finally the author discussed how radix sort can be used to accelerate ray tracing [24].

Gluck *et al* introduced a method for fast quadtree construction on the GPU. The level-by-level approach is used to construct a quadtree. Quadtree is used for the spatial segmentation of lidar data points using a grid digital evaluation model (DEM). The author introduced an algorithm which is suitable for quadtree construction using GPU. The suggested algorithm reduces the construction problem of bucket sort [25].

Ye *et al* presented the GPU based sorting algorithm which is GPUMemSort. It achieved the highest performance in sorting. It has consisted two algorithms [26]:

Polok *et al* focused on the implementation of extremely productive sorting routines for the sparse linear algebra operations. Testing results show that the suggested approach outperforms the other similar implementations. Author implementation is bandwidth efficient because sorting rate is achieved by it compare to the theoretical upper bound on memory bandwidth [27].

Mu *et al* described the bitonic sort algorithm in detail and implementation is done on CUDA architecture. The two effective optimization implementation details are conducted at the same time using the characteristics of the GPU which improves the efficiency. The experimental results show that GPU bitonic sort have 20 times more speed up to the CPU quick sort [28].

Xiao *et al* proposed the high performance approximate sort algorithm based on the CUDA parallel computing architecture running on multi-core GPUs. The algorithm divides the input into distribution multiple small intervals. The results showed that approximate sort is two times faster than radix sort and far exceeds all the GPUs-based sorting [29].

Ajdari *et al* described the modification of the Odd-Even sort. The modification of the algorithm consists in the ability to work with the blocks of elements instead working with individual elements. The modification is done using the CUDA technology. The results showed that sorting of integers in CUDA environment is dozens of times faster [30].

Neetu *et al* presented the GPU merge and quick sort. The objective of the paper is to evaluate and analyze the achievement of merge and quick sort using GPU technology. Author also evaluated the parallel time and space complexity of both algorithms using dataset [31].

4. HARDWARE CONFIGURATION

We have run the algorithms on Window 7 32-bit operating system. The Window 7 has Intel® core™ i3 processor 530 @ 2.93 GHz. The system has the GeForce GTX 460 graphic processor with (7 multiprocessors X (48) CUDA cores\MP) = 336 CUDA cores. The maximum threads per multiprocessor are 1536 and 1024 threads per block. The runtime version of CUDA system is 6.0. The global memory used by the system is 768 Mbytes and the total amount of constant memory is 65536 bytes. The total amount of shared memory per block is 49152 bytes. System having the total number of registers available per block is 32768 and warp size is 32. Maximum sizes of each dimension of a block are 1024 x 1024 x 64 and maximum size of each dimension of a grid is 65535 x 65535 x 65535.

5. EXECUTION TIME TESTING OF PARALLEL AND SEQUENTIAL ALGORITHMS

The certification of the sequential and parallel sorting algorithms is done on a dataset [T10I4D100K (.gz)] [32, 33]. The dataset contains the 1010228 items. The four cases has been chosen for certification.

(1) Random with repeated data (Random data)

(2) Reverse sorted with repeated Data (Reverse sorted data)

(3) Sorted with repeated data (Sorted data)

(4) Nearly sorted with repeated data (Nearly sorted data)

Table 1, expressed the execution time in seconds of numerous types of parallel and sequential sorting algorithms using the dataset.

Table 1. Execution time of numerous types of sequential and parallel sorting algorithms in seconds

Sorting algorithms	Random data	Nearly sorted data	Sorted data	Reverse sorted data
Sequential quick sort	1.043904	1.219802	1.26322	72.089548
Parallel quick sort	0.08001152	0.08501333	0.085001309	0.08501365
Sequential count Sort	0.001841	0.00197	0.0019	0.00199
Parallel count sort	0.000001531	0.000001542	0.000001395	0.000001594
Sequential merge sort	0.266	0.235	0.218	0.219
Parallel merge sort	0.000001632	0.000001568	0.000001504	0.0000016
Sequential Odd-Even Sort	2067.263	596.431	577.812	2002.876
Parallel Odd-Even Sort	358.126	337.654	332.017	348.654

In Table 1, the most efficient parallel and sequential sort is count sort. It is because the count sort is based on the key range and also it is non-comparison based sorting algorithm. The range of count sort is taken from 0 to 65565. The Fig 2 to 5 is represented using the values in Table 2. In all the Fig 2 to 5, the *X*-axis represented the different types of sorting algorithms and the *Y*-axis expressed the execution time in seconds.

The Fig 2 and 3 illustrate the execution time comparison of sequential and parallel sorting using random and nearly sorted data. In both the figures, odd-even sort is taking more time in comparison to others. It is because the odd-even sort is the extension of bubble sort. The count sort is the most efficient sort because of the range of key value.

The Fig 4 and 5 illustrate the execution time comparison of sequential and parallel sorting using sorted and reverse sorted data. In both the figures, odd-even sort is taking more time in comparison to others. It is because the odd-even sort is the extension of bubble sort. The count sort is the most efficient sort because of the range of key value.

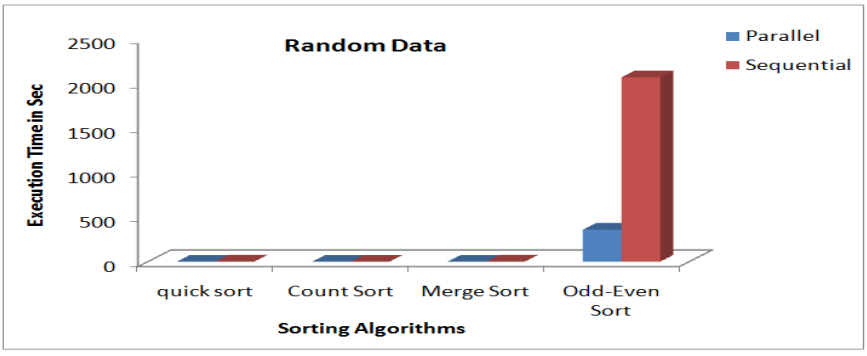

Fig 2: Execution time comparison of various sequential and parallel sorting using random data

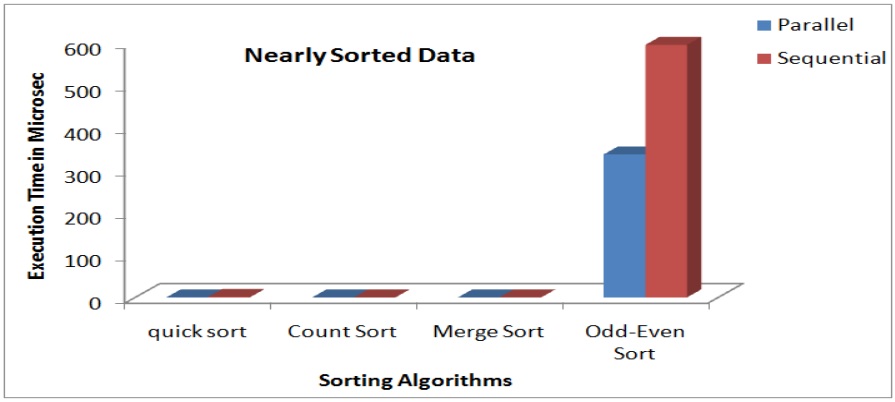

Fig 3: Execution time comparison of various sequential and parallel sorting using nearly sorted data

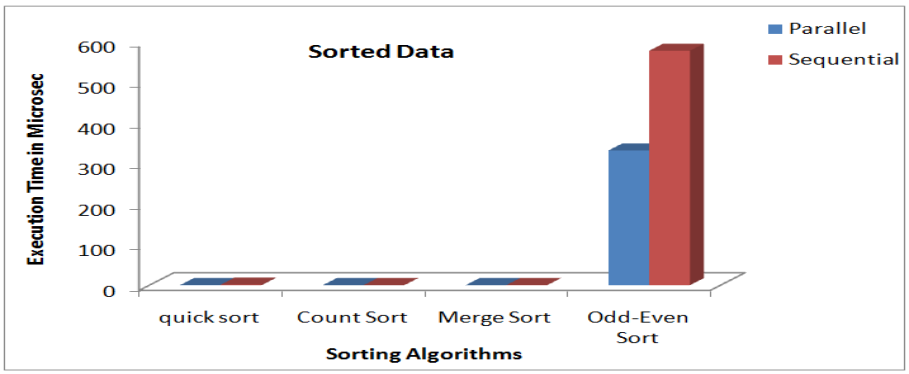

Fig 4: Execution time comparison of various sequential and parallel sorting using sorted data

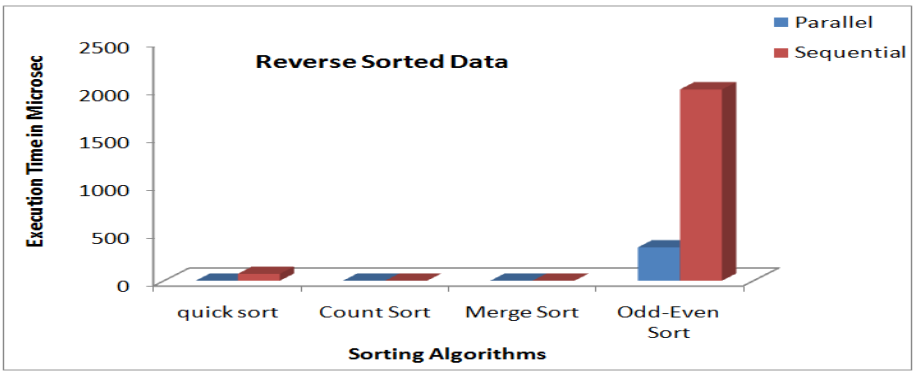

Fig 5: Execution time comparison of various sequential and parallel sorting using reverse sorted data

6. MEASUREMENT OF SPEEDUP

In this section, the speedup of parallel sorting algorithms over the sequential has been shown. The speedup measures the performance gain, which is acquired by parallelizing a given application over sequential application. Table 2 represented the speedup achieved by various parallel sorting algorithms using different types of dataset. By analyzing the Table 2, results shows that merge sort achieved the additional speedup in comparison to others. In the Fig. 6, the *X*-axis expresses the type of datasets and the *Y*-axis shows the speedup acquires by various parallel sorting algorithms.

Table 2: Speedup acquired by parallel sorting algorithms using the various types of datasets

Speedup				
Sorting algorithms	Random data	Nearly sorted data	Sorted data	Reverse sorted data
Quick Sort	13.04692124	14.3483616	14.8611829	847.9761544
Count Sort	1202.482038	1277.561608	1362.007168	1248.431619
Merge Sort	162990.1961	149872.449	144946.8085	136875
Odd-Even Sort	5.772446011	1.766396963	1.740308478	5.744594928

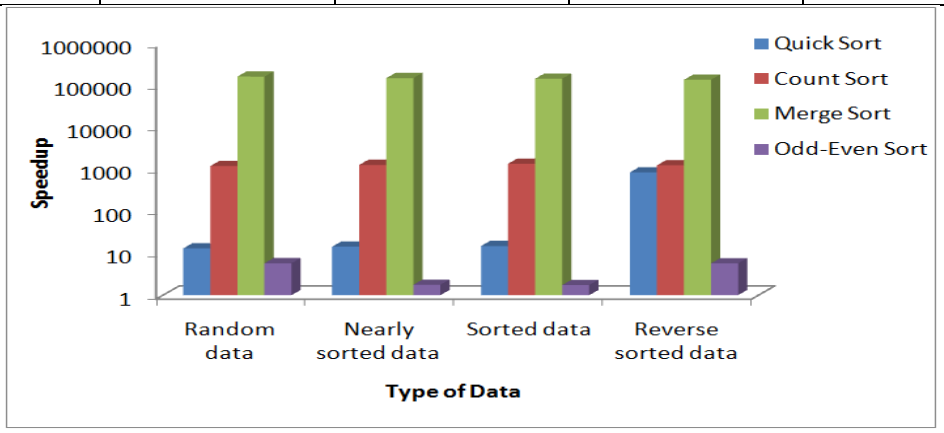

Fig 6: Speedup acquired by various parallel sorting algorithms using various datasets

7. CONCLUSION AND FUTURE WORK

There have been various models developed based on the traditional sorting algorithms like quick sort, merge sort bucket sort and count sort. As in most of the cases the papers have been found based on implementation of the GPU radix sort. The radix sort is based on the key component that is prefix sum. If an efficient way is measured to find the prefix sum, then the efficiency sorting algorithms can increases which are based on prefix sum. The sorting algorithm implementation is established on the description of the data. The description of data affects the sorting process. If the sorting algorithms are designed to handle or take the advantage of the nature of the data, then there will be a huge increase in the performance in that particular case. This may be an overhead for the other cases of the data. The merge sort and quick sort have attracted the interest of various authors, but sample sort claims to be the most efficient in compare to comparison-based sorting algorithms above mentioned.

In this paper some parallel and sequential sorting is tested using the four types of data sets. After comparison, outcome comes that count sort is the most efficient sort in comparison to others. It is because the count sort based on the range of key elements. Speedup is also measured by the parallel sorting algorithms over sequential in which merge sort achieved the additional speedup than others. Four types of sorting algorithms have tested and compared which are merge, quick, count and odd-even sort. In the similar manner the others, sorting algorithms can be tested and compared. The GPU computing using CUDA hardware having the compute capability 2.1 is used to analyze the algorithms. But, if the same algorithms have been used on the hardware having the compute capability 3.0, then it will give an added advantage of unified memory architecture.

The researcher still finds a gap to use the knowledge about the data to implement the sorting algorithm. Future research will refine the performance of sorting algorithms using GPU architecture and Thrust library.

8. ACKNOWLEDGMENTS

This work has been done only for research concern. All experimental results are done in the research lab of Jaypee University of Information Technology, Waknaghat Solan, India.

9. REFERENCES

[1] Greb, Alexander, and Gabriel Zachmann, 2006. GPU-ABiSort: Optimal parallel sorting on stream architectures. IEEE Parallel and Distributed Processing Symposium, IPDPS, 20th International

[2] Inoue, Hiroshi, et al, 2007. AA-sort: A new parallel sorting algorithm for multi-core SIMD processors. Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. IEEE Computer Society.

[3] Sintorn, Erik, and Ulf Assarsson. 2008. Fast parallel GPU-sorting using a hybrid algorithm. Journal of Parallel and Distributed Computing, vol. 68, No. 10, pp. 1381-1388.

[4] Cederman, Daniel, and Philippas Tsigas. 2008. A practical quicksort algorithm for graphics processors. Algorithms-ESA, Springer Berlin Heidelberg, pp. 246-258.

[5] Rożen, T., Krzysztof Boryczko, and Witold Alda. 2008. GPU bucket sort algorithm with applications to nearest-neighbour search.

[6] Baraglia, Ranieri, et al. 2009. Sorting using bitonic network with CUDA. the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR), Boston, USA.

[7] Leischner, Nikolaj, Vitaly Osipov, and Peter Sanders. 2010. GPU sample sort. Parallel & Distributed Processing (IPDPS), International Symposium on IEEE.

[8] Kukunas, Jim, and James Devine. 2009. GPGPU Parallel Merge Sort Algorithm. NVIDIA Technical Report NVR-

[9] Oat, Christopher, Joshua Barczak, and Jeremy Shopf. 2010. Efficient spatial binning on the GPU. SIGGRAPH Asia

[10] Huang, Bonan, Jinlan Gao, and Xiaoming Li. 2009. An empirically optimized radix sort for gpu. Parallel and Distributed Processing with Applications, IEEE International Symposium on.

[11] Ye, Xiaochun, et al. 2010. High performance comparison-based sorting algorithm on many-core GPUs. Parallel & Distributed Processing (IPDPS), IEEE International Symposium on.

[12] Peters, Hagen, Ole Schulz-Hildebrandt, and Norbert Luttenberger. 2010. Fast in-place sorting with cuda based on bitonic sort. Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, pp. 403-410.

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

[14] Satish, Nadathur, et al. 2010. Fast sort on cpus, gpus and intel mic architectures. Intel Labs, pp. 77-80.

[15] Helluy, Philippe. 2011. A portable implementation of the radix sort algorithm in OpenCL.

[16] Krueger, Jens, et al. 2011. Applicability of GPU Computing for Efficient Merge in In-Memory Databases. ADMS@ VLDB.

[17] Mišić, Marko J., and Milo V. Tomašević. 2011. Data sorting using graphics processing units. Telecommunications Forum (TELFOR), 19th. IEEE.

[18] Peters, Hagen, Ole Schulz-Hildebrandt, and Norbert Luttenberger. 2012. A novel sorting algorithm for many-core architectures based on adaptive bitonic sort. Parallel & Distributed Processing Symposium (IPDPS), IEEE 26th International.

[19] Jan, Bilal, et al. 2012. Fast parallel sorting algorithms on GPUs." International Journal of Distributed and Parallel Systems, vol. 3, pp. 107-118.

[20] Munavalli, Seema M. 2012. Efficient Algorithms for Sorting on GPUs.

[21] Thouti, Krishnahari, and S. R. Sathe. 2012. An OpenCL Method of Parallel Sorting Algorithms for GPU Architecture.

[22] Żurek, Dominik, et al. 2013. The comparison of parallel sorting algorithms implemented on different hardware platforms." Computer Science, vol. 14, No. 4, pp. 679-691.

[23] Panwar, Mukul, Monu Kumar, and Sanjay Bhargava. 2014. GPU Matrix Sort (An Efficient Implementation of Merge Sort)." International Journal of Computer Applications, vol. 89, No. 18, pp. 9-11.

[24] Arturo Garcia, Jose Omar Alvizo Flores, Ulises Olivares Pinto, Felix Ramos. 2014. 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.

[25] Gluck, Joshua. 2014. Fast GPGPU Based Quadtree Construction.

[26] Ye, Yin, et al. 2014. GPUMemSort: A High Performance Graphics Co-processors Sorting Algorithm for Large Scale In-Memory Data. Journal on Computing (JoC), vol. 1, No. 2.

[27] Polok, Lukas, Viorela Ila, and Pavel Smrz. 2014. Fast radix sort for sparse linear algebra on GPU. Proceedings of the High Performance Computing Symposium. Society for Computer Simulation International.

[28] Mu, Qi, Liqing Cui, and Yufei Song. 2015. The implementation and optimization of Bitonic sort algorithm based on CUDA. arXiv preprint arXiv: 1506.01446.

[29] Xiao, Jun, Hao Chen, and Jianhua Sun. 2015. High Performance Approximate Sort Algorithm Using GPUs.

[30] Ajdari, Jaumin, et al. 2015. A Version of Parallel Odd-Even Sorting Algorithm Implemented in CUDA Paradigm." International Journal of Computer Science Issues (IJCSI), vol. 12, No. 3.

[31] Neetu Faujdar and Satya Prakash Ghrera. 2015. Performance Evaluation of Merge and Quick Sort using GPU Computing with CUDA. International Journal of Applied Engineering Research, vol. 10, No.18, pp. 39315-39319.

[32] Frequent Itemset Mining Implementations Repository,http://fimi.cs.helsinki.fi accessed on 10/11/2015

[33] Zubair Khan, Neetu Faujdar, et al. 2013. Modified BitApriori Algorithm: An Intelligent Approach for Mining Frequent Item-Set. Proc. Of Int. Conf. on Advance in Signal Processing and Communication, pp. 813-819.