AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

A Hybrid Multi-Phased GPU Sorting Algorithm

[1]Mohammad H. Al Shayeji, [2]Mohammed H Ali and [3]M.D. Samrajesh

[1,2,3]Computer Engineering Department, College of Computing Sciences and Engineering, Kuwait University P.O. Box No: 5969, Safat 13060, Kuwait

ARTICLE INFO

Article history:
Received 30 September 2014
Received in revised form
17 November 2014
Accepted 25 November 2014
Available online 6 December 2014

Keywords:
CUDA, CPU, GPU, Heaps, Parallel Computing, Scheduling, Sorting.

ABSTRACT

As digital data tend to grow exponentially, effective data sorting is an important function in any data processing systems. Sorting algorithms vary depending on its memory usage, number of comparison it makes, its complexity, adaptability etc. GPUs (Graphical Processing Unit) are an essential part of most modern systems. GPUs are very powerful multi-core multithreaded processors. GPUs are specialized in parallel computations and can be used for solving computation intensive problems. Exploiting the parallelism in GPU design and developing parallel sorting algorithms can make sorting more efficient. In this paper we present a parallel algorithm for sorting large set of data on GPU. The proposed Hybrid Multi-Phased GPU sorting algorithm (HMP) exploits the parallelism of modern GPU architecture; HMP is a hybrid algorithm of heap-sort and bitonic-sort algorithms. The proposed algorithm consists of 3 main phases the splitting phase, sorting phase, and merge phase. Our evaluation and discussions shows that the proposed algorithm has less execution time when compared to bitonic-sort, merge-sort and even-odd sort under different types of datasets.

To Cite This Article: Mohammad H. Al Shayeji, Mohammed H Ali and M.D. Samrajesh, A Hybrid Multi-Phased GPU Sorting Algorithm. ***Aust. J. Basic & Appl. Sci.,*** *8(23): 315-322, 2014*

INTRODUCTION

The exponential growth of digital data makes data cataloguing, processing, and retrieving more challenging (Lee, Victor W., *et al*., 2010). Sorting massive data efficiently has been among the main computational challenges in data processing for decades (Purcell, T.J., *et al*., 2003). Most researches on sorting focus on optimizing certain aspects of the problem such as time complexity and efficient memory management for large dataset (Albutiu, Martina-Cezara, *et al*., 2012; Merrill, Duane G., and Andrew S. Grimshaw, 2010). Recently, many applications have shifted processing from CPU (Central Processing Unit) to GPU (Graphics Processing Unit). However, GPU architecture differs from that of CPU in a number of aspects. Generally, GPU has more cores than most multicore processors available today. Moreover, GPUs can handle thousands of threads simultaneously (Hou, Qiming, *et al*., 2011).

Today, higher hardware capabilities have reduced the sorting time for smaller dataset. However, the sorting time for larger dataset using GPUs is significantly higher and designing efficient sorting algorithms is still a challenge (Capannini, Gabriele, FabrizioSilvestri, and RanieriBaraglia, 2012).

In this paper, we present a parallel algorithm for sorting large set of data on GPU. The proposed Hybrid Multi-Phased GPU sorting algorithm (HMP) is a hybrid algorithm of heap-sort and bitonic-sort algorithms and exploits the parallelism of modern GPU architecture. The proposed algorithm consists of three main phases the splitting phase, sorting phase, and merge phase. Heap-sort is done locally in the on-chip shared memory, to reduce number of scatters to global memory and maximize the coherency property.

Our proposed algorithm is implemented in CUDA (Compute Unified Device Architecture) by C99 code that uses NVIDIA GeForce video card. We performed experiments and evaluated the execution time of the proposed algorithm under 16-bit integer and 32-bit integer data sets. Moreover, we analyzed the performance under different data sizes. Our evaluation result shows that the proposed Hybrid Multi-Phased GPU sorting algorithm (HMP) sorts large dataset with minimum execution time when compared to bitonic-sort, merge-sort and even-odd sort.

The rest of paper is organized as follows: Section 2 describes the related works. Section 3 provides a background of different sorting algorithms based on which the proposed algorithm is built. Section 4 presents the proposed Hybrid Multi-Phased GPU sorting algorithm. The evaluation and discussions are presented in Section 5 and finally our conclusion and future work are presented in Section 6.

Corresponding Author: Mohammad H. Al Shayeji, Computer Engineering Department, College of Computing Sciences and Engineering, Kuwait University
P.O. Box No: 5969, Safat 13060, Kuwait alshayej@eng.kuniv.edu.kw

Related Work:

There are numerous parallel sorting techniques implemented for both CPU and GPU. Recently, researchers focused on harnessing the power of GPU due to its enormous processing capacity that is often under-utilized. Numerous GPU sorting algorithms use hardware specification of previous generations GPUs, hence they are not relatively efficient (Capannini, Gabriele, FabrizioSilvestri, and RanieriBaraglia, 2012).

A merge sort algorithm based on parallel data architectures for GPU sorting is presented in. The algorithm sorts photons for illumination of realistic images into a spatial data structure providing an efficient search mechanism. It uses breadth-first photon tracing method to distribute photons. However, their implementation is compute-bound rather than bandwidth-bound and the results generated are below the theoretical best of the target architecture.

A fast sort implementation of bitonic-sort for GPU using CUDA is presented in (Peters, H., *et al*., 2009). The algorithm attempts to reduce memory accesses time by having minimum number of access to the memory. It uses an effective instruction dispatch and caching mechanism to reduce the overall sorting time. However, their evaluation results show that the experiments are not able to demonstrate its advantages considering all the aspects of the proposed model.

A comparison-based merge sort is implemented based on divide and conquers approach in (Satish, N., *et al*., 2009). Initially it divides and locally sort a set of sequences, later, each block merges two adjacent sequences until the entire sequence is sorted. Parallelism is maintained and each element is assigned to a thread. However, the level of granularity implementation is too high as each thread handles one element and this has higher overhead.

An odd-even merge sorting algorithm is presented in (NVIDIA CUDA Programming Guide, 2008). It is based on a merge algorithm that combines two sorted halves of a sequence to a fully sorted sequence. However, this algorithm is not data-dependent and cannot be used for a general purpose sorting.

Bitonic sort based parallel merge sort is presented in (Blelloch, Guy E., *et al*., 1991), the sorting is based on merging "bitonic" sequences. A bitonic sequence is one that increases monotonically and then decreases monotonically. The algorithm finally merges both sub sequences in order to produce a globally sorted sequence. However, the input sequence has to pass the full sorting cycle in order to get sorted, thus incurring additional overhead.

A simple, fast parallel implementation of quicksort is presented in (Tsigas, Philippas and Yi Zhang, 2003). This divides each sequence of data that needs to be sorted into blocks that are dynamically assigned to available processors. It is a fine-grain extension of quicksort. However, this approach requires extensive use of synchronization that in turn makes it too expensive to use on graphics processors.

A parallel sorting algorithm using stream programming model for GPUs based on adaptive bitonic sorting is presented in (Greß, A. and G. Zachmann, 2006). In the stream programming model program, structure is described by streams of data passing through computation kernels. This approach can also be implemented on future stream architectures with the condition that the memory should have of a single contiguous memory block. However, the algorithm implementation assumes that the length of the input data is only in power of two.

Background:

Sorting Algorithms:

Sorting algorithms vary based on various factors including the method it uses to arrange data and compare. They can be classified based on the memory usage, number of comparison, complexity, adaptability and much more. The efficiency of the algorithm and performance can be significantly improved by exploiting GPU parallelism. Parallel algorithms are exceptionally effective in solving sorting problems. Parallel sorting algorithms are designed to minimize the dependencies between various elements. It can be categorized as follows (Quan Yang; Zhihui Du; Sen Zhang, 2012):

Splitter-based algorithms:

This type of algorithms partitions the data set into several disjoint chunks as a first phase of the algorithm. The second phase merges the distributed chunks. Examples of this kind of sort include bucket sort, flash-sort, and sample sort.

Merge-based algorithms:

This method uses splitter-based algorithms in their first phase of sub-sequences sorting, and then merge the sorted data to completely ordered elements. Additionally, varied principles for merge is used and this includes bitonic merge, odd-even merge, thrust merge, etc.

Others:

In this category the parallelization of the traditional versions of sorting algorithms are used and it applies the basic approach of divide and conquer method.

Australian Journal of Basic and Applied Sciences, 8(23) Special 2014, Pages: 315-322

GPU and CUDA Programming Model:

Today's GPUs are massively multithreaded processors with a SIMT (Single Instruction Multiple Thread). NVIDIA GPUs consist of a stream of processors that have hundreds of threads organized in blocks. Each block has its own Shared Memory(SM), which can contact global memory of the GPU. Threads are mapped to SMs through warps. CUDA (Lindholm, E., *et al*., 2008; Nickolls, J., *et al*., 2008) is a parallel programming platform developed by NVIDIA to harness the power of modern GPUs by providing C/C++ APIs (Purcell, T.J., *et al*., 2003). Latest CUDA programming model has introduced a unified memory address mechanism (Shi, H. and J. Schaeffer, 1992) to unify the global memory of the GPU with the CPU memory.

Heaps:

Heaps are binary trees. Generally, there are two types of heaps 1) Max-Heap 2) Min-Heap. Heaps have many applications; the most important one is using heaps in priority queue. Fig. 1 shows Max-Heapify pseudo-code. Heaps are used for sorting, heap-sort is a popular algorithm with complexity *O (nlog n).*

Bitonic-Sort:

Bitonic sorting algorithms are considered the most suitable sorting algorithms for GPU based system. The sequence of comparisons is data independent. Most enhancements in sorting algorithms under GPU platform are based on bitonic-sort (Peters, H., *et al*., 2009). The sorting algorithm works with data of size $n = 2^k$, k is the number of steps. For sorting an arbitrary list, the algorithm involves 2 steps. First, generating a bitonic sequence by converting arbitrary sequences to bitonic ones. Second, the bitonic sequence is sorted. A bitonic sequence is a sequence that either monotonically decreases or monotonically increases.

Max-HeapifyAlgorithm

```
Max-Heapify (A, i)
l←2i+1
r←2i+2
if l≤ HeapSize and A[l] > A[i]
    then largest← r
    else largest← i
if r≤ HeapSize and A[r] > A[largest]
    then largest← r
if largest≠ i
    then exchange A[largest]↔A[i]
         Max-Heapify (A, largest)
```

Fig. 1: Max-Heapify Algorithm

Heap Sort using Max-Heapify

```
HeapSort (A)
        for i ← length A down to 1
                do exchange A[1]↔A[i]
        HeapSize← HeapSize - 1
        Max-Heapify (A,0)
        end for
```

Fig. 2: Heap Sort using Max-Heapify

e.g. 1 2 3 4 5 4 3 1 is a bitonic sort. The algorithm takes exactly *k* steps for $n = 2^k$. The sorting phase in step 2 is comparing each element *i* in *A* with *A[n/2*] until *i=n/2 -1* with *A[n-1]*.Bitonic sort has $O\ (n \log n^2)$ comparators in every step n/2 operations (NVIDIA CUDA C Programming Guide, 2014). An *n*-element bitonic sequence required $k = O\ (\log n)$. Sorting arbitrary sequences on *n*-elements take *k* steps where each sub-steps take $O\ (\log^2 n)$.

Proposed Gpu Sorting Algorithm:

The proposed sorting algorithm consists of 3 main phases first the splitting phase, second the sorting phase, and finally the merge phase. Heap-sort is done locally in the on chip shared memory, to reduce number of scatters to global memory and maximize the coherency property.

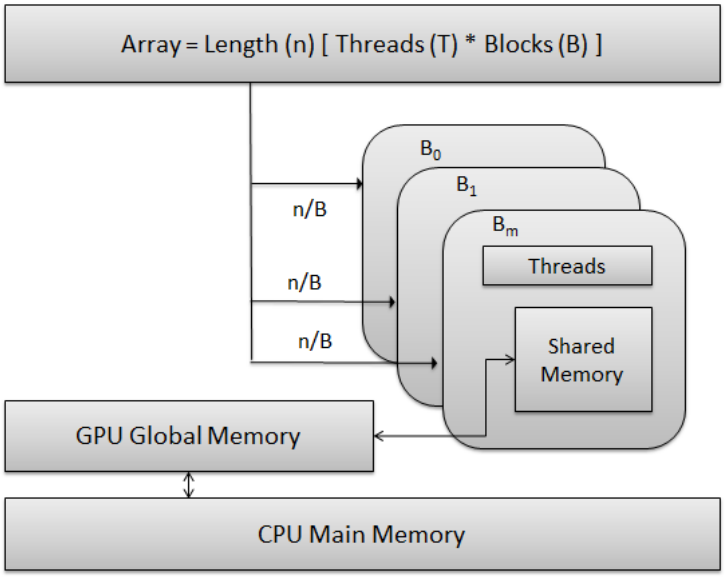

Fig. 3: Block diagram of the Splitting Phase

The splitting phase:

In this phase, the array is partitioned into number sub-sequences called blocks, each block are of equal size and holds different set of the original array elements. The block set is represented as warps: warps contain threads. The size of the warps, the number of blocks and number of threads used depend on the problem and the size of the data. Fig. 3 shows the overall view of the splitting phase. The main aim of splitting the array into chunks of blocks of equal size *(n)* is to fit the chunks of the array in GPU memory such that the GPU resources are used effectively.

Sorting chunks of data

In the proposed HMP algorithm, first the sub-sequences are sorted by heap-sort using both type of heaps, Min-Heap and Max-Heap Refer Fig.2. The numbers of comparisons made are less, as bitonic-sort is used to perform sorting. The algorithm distributes the sorted arrays. The first heap after sorting will produce an ascending sorted chunk while the second one will produce a descending sorted chunk. In the end of this phase a sorted sequence (bitonic sequences) is produced. Building a heap complexity is *O (n)*, while Heapify complexity is *O (log n)* for each block and for heap-sort the overall complexity for each block is *O (n log n).*

Merging Sequences:

Sequences are merged using bitonic-sort, more sequences are sorted simultaneously based on number of blocks and each number of threads per block. In this step bitonic-sort is more efficient, since the sequences are already in bitonic fashion. In addition, bitonic sort uses minimal number of comparisons to swap between elements. The ordered sequences are merged together with worst-case complexity $O\ (n \log_2 n)$.The complexity of the algorithm for sorting elements is *O (log n)*.

Fig. 5 shows an illustrative numerical example of sorting 10 integer numbers. Initially, the splitting phase partitions the data into number sub-sequences called blocks, each block is of size 2, and each block holds 2 elements. Next, the sub-sequences are sorted by heap-sort using both type of heaps the Min-Heap and Max-Heap. The first heap after sorting produces elements sorted in ascending order while the second one produces elements sorted in descending order. Refer Fig. 5 *(*)*, we have 2 sequences; these sequences are generated by max heap and min-heap. The first sequence *(A)* is in increasing order and second sequence *(B)* is in decreasing

order, thus the entire sequence is in bitonic sequence. Now, the sequences are merged using bitonic-sort and, sorted simultaneously the numbers underlined in the figure represent the comparison and swapping of elements and finally the entire sequence is sorted as shown in Fig. 5.

Algorithm HeapBitonic Sort

```
HeapBitonicSort (A)
ThreadPerBlock← 32
Blocks ← Size/ ThreadPerBlock
for Blocks down to 0
        if Block % 2 == 0
                thenHeapSort with Max-Heap
                elseHeapSort with Min-Heap
    end for
store sorted chunks t o A again
ThreadPerBlock← 2*ThreadPerBlock
BitonicSort (A, ThreadPerBlock)
```

Fig. 4: Heap Sort using Max-Heapify

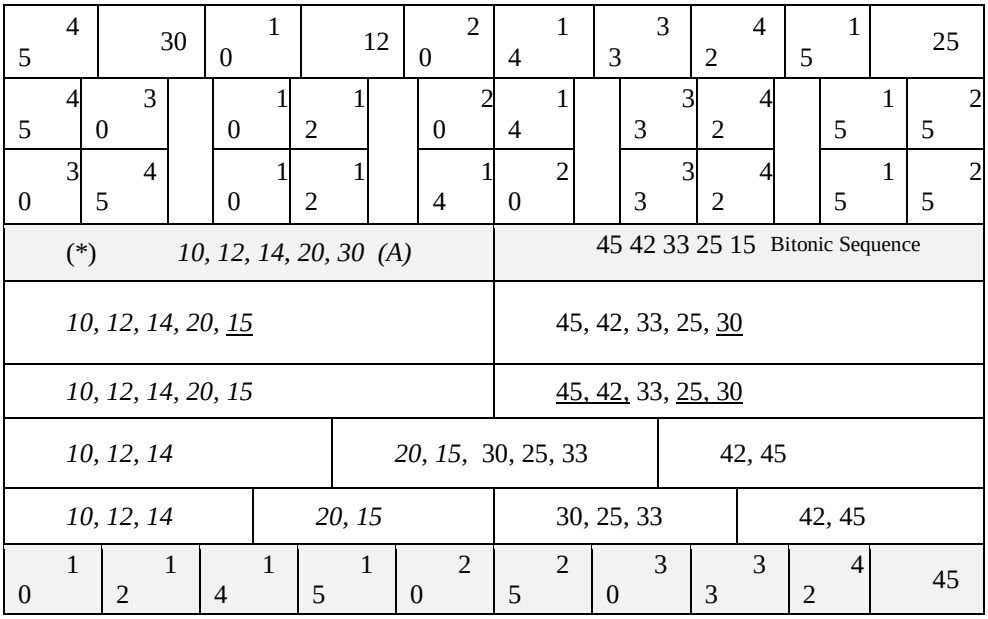

Fig. 5: Illustrative numerical example

RESULTS AND DISCUSSIONS

Experimental Setup:

All experiments are done using system consisting of NVIDIA GeForce GT 750M video graphic card. It supports PCIe v. 3. The core speed is 835 MHz and has Kepler GK107 design architecture. The memory is 2048 MB with speed 1000MHz. Memory type is GDDR5 with 64GB/sec memory bandwidth. The host machine used is Apple MacBook Pro Retina, 15-inch, with 2.3 GHz Intel core i7 CPU and 16 GB 1600 MHz DDR3 main memory.

Implementation and Parameter:

Hybrid Multi-Phased (HMP) GPU sorting algorithm is implemented using CUDA programming platform. The proposed algorithm is compared with high performance GPU sorting algorithms (Ye, Xiaochun, *et al*., 2010), such as the merge sort in (Satish, N., *et al*., 2009), the bitonic-sort and the odd-even merge sort in (NVIDIA CUDA Programming Guide, 2008). The above algorithms are selected for comparison since their functionality (Batcher, Kenneth E., 1968) closely resembles the proposed HMP. We use16-bit integers and 32-

bit integers dataset to perform our comparative evaluation. Our comparatives evaluation strategy is based on (Lee, Victor W., *et al*., 2010).

Discussions:

We evaluated the proposed HMP GPU sorting algorithm using 16-bit integer and 32-bit integer dataset. Fig. 6 shows the execution time of 16-bit integer sorting. The HMP outperforms other algorithms as it takes least execution time. Further observation shows that the execution time of HMP is lower since HMP's data sequences are already in bitonic fashion during the sorting process and does less data comparison and uses effective thread management.

Fig. 7 shows the execution time of 32-bit integer GPU sorting. Initially, the execution time of the proposed HMP algorithm is higher for smaller data size. However, when the data size increases it execution time is lower when compared to other algorithms; the additional time taken by HMP at lower data size is due to the overhead of partitioning data into sub-sequences and merging.

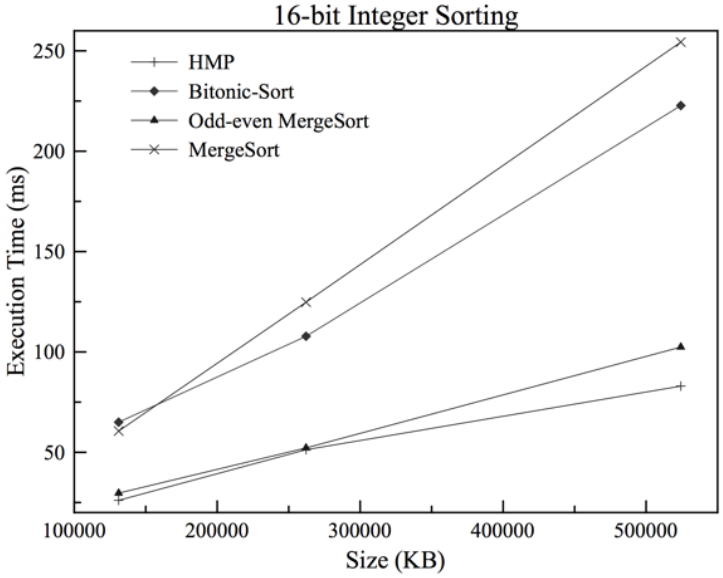

Fig. 6: 16-bit Integer Sorting

Fig. 6 and Fig 7 shows that HMP and odd-even merge sort algorithms have comparable performance due to their similar thread allotment and management, likewise in the case of merge sort and bitonic sort algorithms. Additionally, during the process of data sequences merging, sub-sequences of each thread per block incur additional overhead and thus an increase in execution time. Moreover, the proposed algorithm tends to have least execution time in both cases of integers since HMP makes less number of comparison and uses on chip shared memory to reduce number of data scatters to global memory.

Conclusion And Future Work:

We presented and implemented a Hybrid Multi-Phased (HMP) GPU sorting algorithm (HMP) using CUDA programming platform. The experiments show that the proposed algorithm outperforms bitonic-sort, merge-sort and even-odd sort GPU algorithms for 16-bit and 32-bit sorting integers sorting. The evaluation result shows that HMP is efficient for larger dataset elements. Heap-sort is done locally in the on chip shared memory to

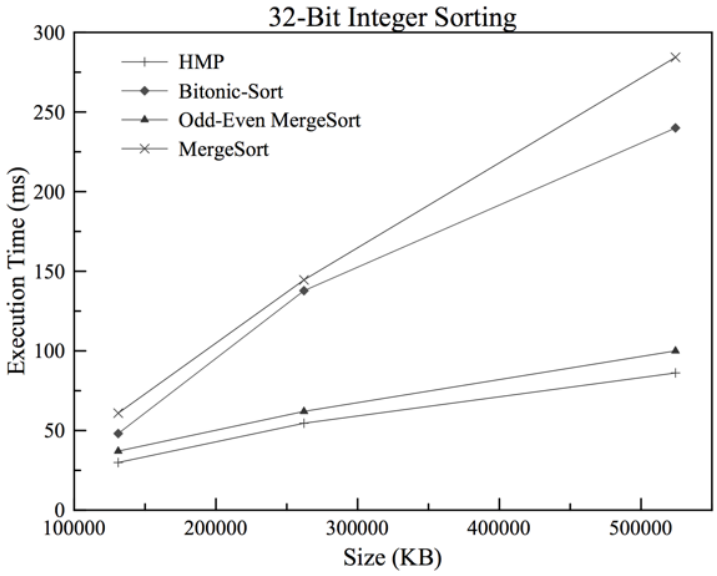

Fig. 7: 32-bit Integer Sorting

reduce number of scatters to global memory and maximize the coherency property. As a part of future work we plan to test he algorithm under different GPU architecture such as Quadro, Tesla and Maxwell and use other data types such as float and double.

REFERENCES

Albutiu, Martina-Cezara, Alfons Kemper and Thomas Neumann, 2012. "Massively parallel sort-merge joins in main memory multi-core database systems." Proceedings of the VLDB Endowment., 5(10): 1064-1075.

Batcher, Kenneth E., 1968. "Sorting networks and their applications." Proceedings of the April 30--May 2, 1968, spring joint computer conference. ACM.

Blelloch, Guy E., *et al.*, 1991. "A comparison of sorting algorithms for the connection machine CM-2." Proceedings of the third annual ACM symposium on Parallel algorithms and architectures. ACM.

Capannini, Gabriele, FabrizioSilvestri, and RanieriBaraglia, 2012. "Sorting on GPUs for large scale datasets: A thorough comparison." Information Processing & Management., 48(5): 903-917.

Greß, A. and G. Zachmann, 2006. "Gpu-abisort: Optimal parallel sorting on stream architectures," in Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS '06), p: 45.

Hou, Qiming, *et al.*, 2011. "Memory-scalable GPU spatial hierarchy construction." Visualization and Computer Graphics, IEEE Transactions., 17(4): 466-474.

Jan, B., B. Montrucchio, C. Ragusa, F.G. Khan and O. Khan, 2012. "A Fast Parallel Sorting Algorithms on GPUs", International Journal of Distributed and Parallel Systems (IJDPS), 3.

Lee, Victor W., *et al.*, 2010. "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU." ACM SIGARCH Computer Architecture News. 38(3).

Lindholm, E., J. Nickolls, S. Oberman and J. Montrym, 2008. NVIDIA Tesla: A unified graphs and computing architecture. IEEE Micro, 28(2): 39-55.

Merrill, Duane G., and Andrew S. Grimshaw, 2010. "Revisiting sorting for GPGPU stream architectures." Proceedings of the 19th international conference on Parallel architectures and compilation techniques. ACM,

Nickolls, J., I. Buck, M. Garland and K. Skadron, 2008."Scalable parallel programming with CUDA," Queue, 6(2): 40-53.

NVIDIA CUDA C Programming Guide, 2014. NVIDIA Corporation, Design Guide, version 6.0.

NVIDIA CUDA Programming Guide, 2008. NVIDIA Corporation, Jun. version 2.0.

Peters, H., O. Schulz-Hildebrandt, and N. Luttenberger, 2009. "Fast in-place sorting with cuda based on bitonic sort" . In To appear in: PPAM09: Proceedings of the International Conference on Parallel Processing and Applied Mathematics.

Purcell, T.J., C. Donner, M. Cammarano, H.W. Jensen and P. Hanrahan, 2003. Photon mapping on programmable graphics hardware. In Proceedings of the ACM SIGGRAPH Conference on Graphics Hardware, pages 41–50. Eurographics Association.

Quan Yang; Zhihui Du; Sen Zhang, 2012. "Optimized GPU Sorting Algorithms on Special Input Distributions," Distributed Computing and Applications to Business, Engineering & Science (DCABES), 2012

11th International Symposium on, pp: 57,61, 19-22 doi: 10.1109/DCABES.2012.57

Satish, N., M. Harris and M. Garland, 2009. "Designing efficient sorting algorithms for many core GPUs," in Proc. 23rd IEEE International Symposium on Parallel and Distributed Processing (23rd IPDPS'09), (Rome, Italy), pp. 1–10, IEEE Computer Society.

Shi, H. and J. Schaeffer, 1992. Parallel Sorting by Regular sampling, Journal of Parallel and Distributed Computing., 14: 361-372.

Tsigas, Philippas and Yi Zhang, 2003. "A simple, fast parallel implementation of quicksort and its performance evaluation on SUN enterprise 10000." Parallel, Distributed and Network-Based Processing, 2003. Proceedings. Eleventh Euromicro Conference on. IEEE.

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