Contemporary Engineering Sciences, Vol. 5, 2012, no. 10, 455 - 462

Number of Processors Needed for Batcher-Bitonic, Radix and Quick Sorting Networks Methods

Ashraf Abdel-Karim Abu-Ein

Department of Computer Engineering, Faculty of Engineering Technology
Al-Balqa' Applied University
P.O. Box, 15008 Amman 11134, Jordan
ashraf.abuain@fet.edu.jo

Abstract

This paper demonstrated different networking sorting techniques and comparing between them. This paper also aims to calculate number of processors needed to make such sorting methods. The efficiency of such sorting methods is differ, it depends on time needed to execute some given tasks, some efficiency measures are developed or derived to make the comparison between such methods enabled. As the number of processors increases the time decreases. A suggested method which is consider to be a bitonic modified method depends on Fractal geometry principles that divide any collection of networks items into smaller and smaller groups then make sorting between the most small groups and go up to bigger one to get sorted items at the end of the time. The modified method required less processors to make same network sorting.

Keywords: Batcher sorting, Quick and Radix Sorting, Efficiency, processors, networks

1. Introduction

Sorting things in all times is an important issue, sorting may starting from simple things to more complicated problems, someone may sorting similar things from a collection of mixed things to choosing or sorting things from a memory of machine

containing a huge stored data or information. Since 1950's many sorting algorithms had been studied and investigated, started by Knuth, who discussed the Quicksort, Radix sorts, and Radix-Exchange sorts. But from 1968 the parallel sorting algorithms appeared such as Batcher sorting algorithm which is considered as bitonic algorithm which requires n processors of $O(\log^2 n)$. figure 1and shows the bitonic sorter block diagram, while figure 3 shows the iterative rule for odd-even merging networks.[1, 2, 7, 8]

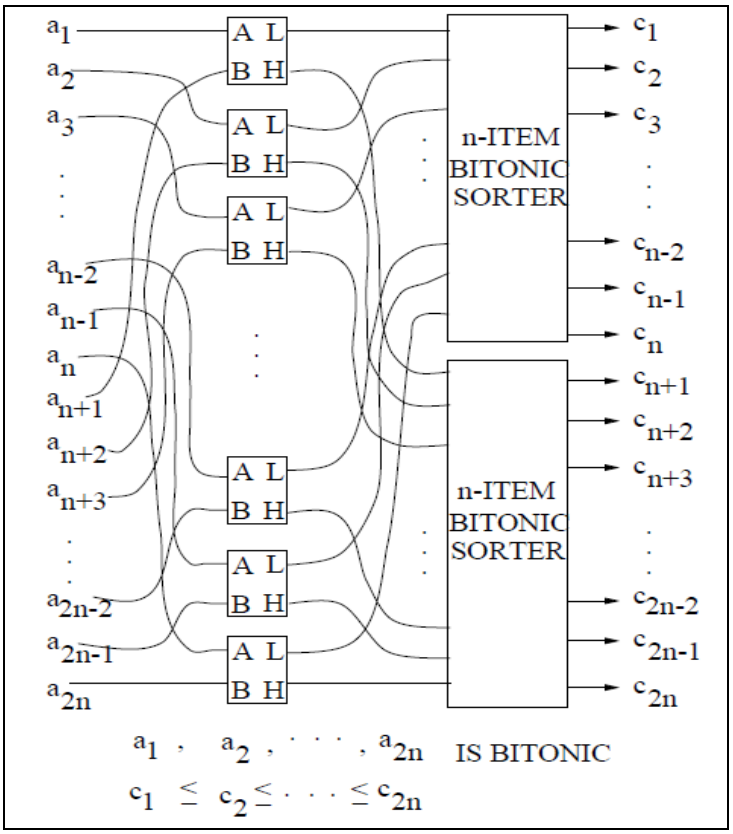

Figure 1: iterative rule for bitonic sorters,[6].

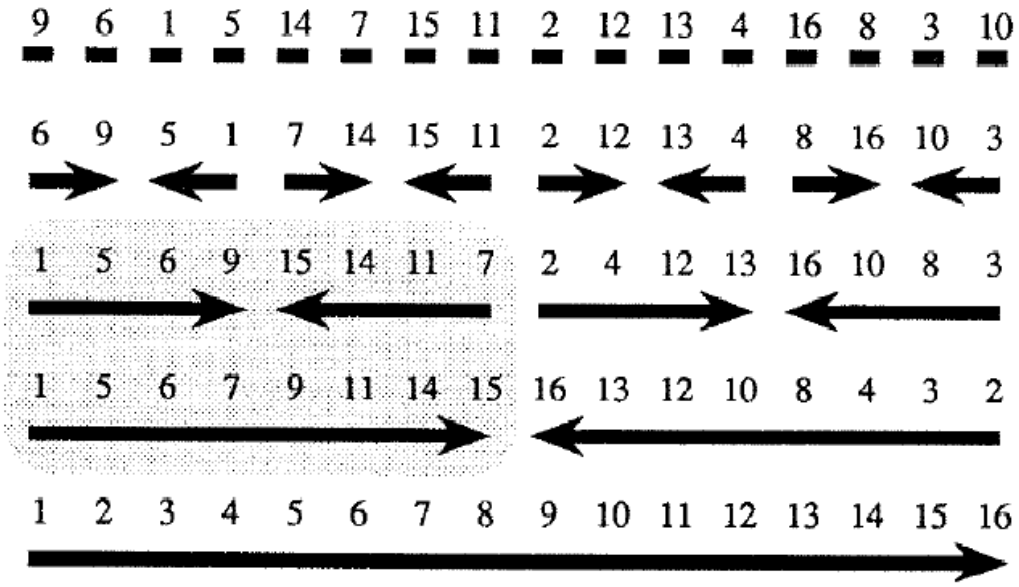

Figure 2 bitonic sorting operations

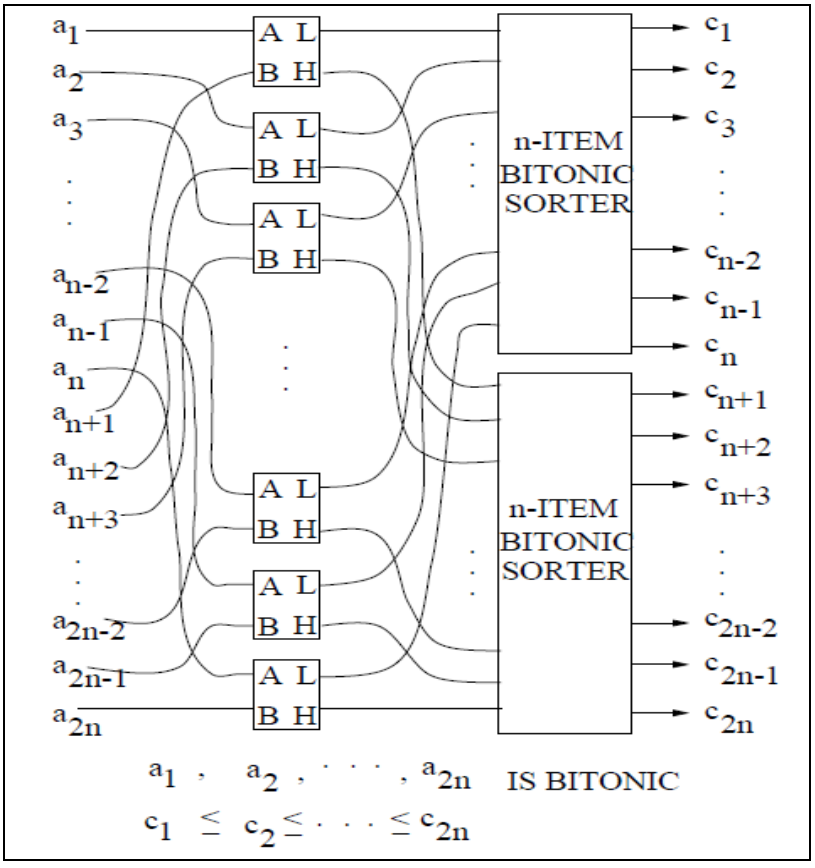

Figure 3: iterative rule for odd-even merging networks,[6].

Many sorting methods had been discussed and analyzed by many articles, as an example parallel algorithms on real parallel machines were discussed by G. E. Blelloch, et al. 1998, who developed a methodology for predicting the performance of such sorting methods. His methodology consists of two steps. The first step is to characterize a machine by enumerating the primitive operations that it is capable of performing along with the cost of each operation. The second step is to analyze an algorithm by making a precise count of the number of times the algorithm performs each type of operation. Three most promising algorithms are selected: Batcher's bitonic sort, a parallel radix sort, and a sample sort similar to Reif and Valiant's flash sort. They also analyzed the three algorithms in detail and discusses the issues that led to their particular implementations. On the CM-2 the predicted performance of the algorithms closely matches the observed performance, and hence their methodology can be used to tune the algorithms for optimal performance.

Brandon Dixon, et al. 1995, they studied the sorting performance of the CRAY T3D on a variety of sorting tasks. The problems range from that of sorting one word per processor to sorting the entire memory of the machine, they gave an efficient algorithms for each case. Linh H. et al. 2009, in this paper they presented a hardware-optimized parallel implementation of the radix sort algorithm that results in a significant speed up over existing sorting implementations. They outperformed all known GPU based sorting systems by about a factor of two and eliminate restrictions on the sorting key space. This makes the algorithm not only the fastest, but also the first general GPU sorting solution.

Nadathur S. et al. 2009, they described the design of high-performance parallel radix sort and merge sort routines for many core GPUs, taking advantage of the full programmability offered by CUDA. The suggested radix sort is the fastest GPU sort and the merge sort is the fastest comparison-based sort reported in the literature. Radix sort is up to 4 times faster than the graphics-based GPU Sort and greater than 2 times faster than other CUDA-based radix sorts. It is also 23% faster, on average, than even a very carefully optimized multi core CPU sorting routine. They carefully design an algorithms to expose substantial fine-grained parallelism and decompose the computation into independent tasks that perform minimal global communication.

Gabriele C. et al. 2011, In this paper, they compared different algorithms for sorting integers on stream multiprocessors and they discussed their viability on large datasets (such as those managed by search engines). They designed an optimized version of sorting network in the K-model, a novel computational model designed to consider all the important features of many-core architectures.

A summery for some sorting algorithms is shown in table 1a and b below.

This paper comparing between the existed methods and a new suggested method which depends on Fractal geometry principles that divide any collection of networks items into smaller and smaller groups then make sorting between the most small groups and go up to bigger one to get sorted items at the end of the time.

Table 1 a, and b network sorting algorithms comparison[7, 9,10]

Name	Best	Average	Worst	Memory	Stable	Method
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends	Partitioning
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends; worst case is n	Yes	Merging
In-place Merge sort	—	—	$n \left(\log n\right)^2$	1	Yes	Merging
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection
Insertion sort	n	n^2	n^2	1	Yes	Insertion
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection
Selection sort	n^2	n^2	n^2	1	No	Selection
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No	Insertion
Bubble sort	n	n^2	n^2	1	Yes	Exchanging
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes	Insertion
Cycle sort	—	n^2	n^2	1	No	Insertion
Library sort	—	$n \log n$	n^2	n	Yes	Insertion
Patience sorting	—	—	$n \log n$	n	No	Insertion & Selection
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection
Strand sort	n	n^2	n^2	n	Yes	Selection
Tournament sort	—	$n \log n$	$n \log n$			Selection

2. Results and Discussion

Time required to execute any sorting operation is critical to judge if the sorting operation as the time decrease the efficiency increase. The time is used to calculate the number of processors needed to execute some sorting operation by different types of sorting techniques, the number of processors are used as follows[2, 3, 4, 6]:
-for bitonic sorting:

$$P = \frac{(T + nQ) + T[(1 + \frac{nQ}{T})^2 - \frac{4nQ}{2T}(\frac{nQ}{2T} - \frac{5nA}{2T} Log^2 n)]^{0.5}}{nQ} \quad (1)$$

-merge sorting

$$P = 0.5 + 0.5\sqrt{1 - 4(1 - Qd + 5Ad)} \quad (2)$$

-radix sorting

$$P = \frac{bRn}{rT_{radix} - bT_{rank}} \quad (3)$$

-modified bitonic method-fractal method
This method depends on using fractal principles in making network processors the number of processors may equal to the half of the inputs. The general equation to calculate number of processors in the modified bitonic method can be calculated by:

$$P = \frac{(T_{mb} + nQ) + T_{mb}[(1 + \frac{nQ}{T_{mb}})^2 - \frac{4nQ}{2T_{mb}}(\frac{nQ}{2T_{mb}} - \frac{5nA}{2T_{mb}} Log^2 n)]^{0.5}}{nQ} \quad (4)$$

The time needed for such operations can be given as:

$$T_{mb} = \frac{T_b}{F^G} \quad (5)$$

Where: P: is the number of processors, n: number of inputs, Tb: is the time of bitonic sorting method, Tmb: is the time of modified bitonic sorting method, Q: is the swapping time, R: is the sending time, A: is algorithmic time which is constant about 1 microseconds, T: is the time, r: is the bit value=11, d: the step number, d=1,……, log n, b: is the bit-key =64. F: is the fractal rank, G: number of inputs in each fractal rank.

Figure 4 shows a comparison between the number of processors needed for each last sorting methods.

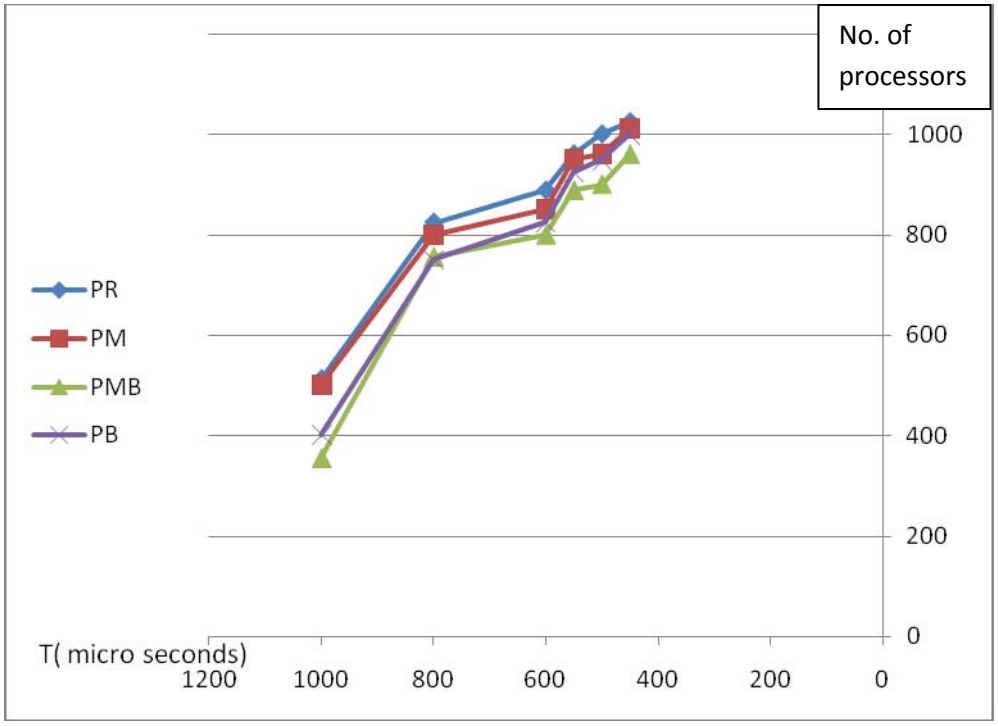

Figure (4) number of processors vs. time needed to make network sorting algorithms. Where PR: Radix method processors, PM: merge method, PB: bitonic method, and PMB: modified bitonic method.

From the figure it can be noticed that the number of processors increases as the sorting time decreases, also the modified bitonic method-fractal method- needs less processors comparing with other methods.

3. Conclusion

Modified bitonic method is a suggested method depends on Fractal geometry principles that divide any collection of networks items into smaller and smaller groups then make sorting between the most small groups and go up to bigger one to get sorted items at the end of the time. The time needed to make sorting by this suggested method is less comparing with other methods and hence the number of processors is reduced too.

References

[1] *Brandon Dixon*, and *John Swallow*, 1995, Sorting on the Cray T3D, CUG 1995 Spring *Proceedings*.

[2] G. E. Blelloch, B. M. Maggs, S. J. Smith,4 and M. Zagha, 1998, "An Experimental Analysis of Parallel Sorting Algorithms", Theory Comput. Systems 31, 135–167 (1998)
[3] Linh Ha, Jens Krüger, Cláudio T. Silva, 2009, Fast 4-way parallel radix sorting on GPUs, COMPUTER GRAPHICS Forum (2/2009).
[4] Nadathur Satish, Mark Harris Michael Garland, 2009, Designing Efficient Sorting Algorithms for Many core GPUs Proc. 23rd IEEE International Parallel and Distributed Processing Symposium, May 2009.
[5] Gabriele Capannini, Fabrizio Silvestri, Ranieri Baraglia, 2011, Sorting on GPUs for large scale datasets: A thorough comparison, Information Processing and Management xxx (2011) xxx–xxx
[6] J. L. Goldstein, and S. W. Leibholz, 1967, On the synthesis of signal switching networks with transient blocking IEEE Transactions EC-16 5 637-641 1967.
[7] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. Journal of the Association for Computing Machinery, 29(3):642–667, 1982.
[8] C. G. Plaxton. Efficient computation on sparse interconnection networks. Technical Report STAN-CS- 89-1283, Department of Computer Science, Stanford University, September 1989.
[9] GRESS A., ZACHMANN G.: GPU-ABiSort: Optimal parallel sorting on stream architectures. In Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS) (Rhodes Island, Greece, 25–29 April 2006).
[10] [8] M. Dowd , Y. Perl , L. Rudolph , M. Saks, “The periodic balanced sorting network”, Journal of the ACM (JACM), v.36 n.4, p.738-757, Oct. 1989

Received: August, 2012