International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

IJIEMR Transactions, online available on 3rd Aug 2019. Link :http://www.ijiemr.org/downloads.php?vol=Volume-08&issue=ISSUE-08

Title **EFFICIENT COMPARISON-FREE SORTING ALGORITHM FOR HIGH EFFICIENCY IN O(N)**

Volume 08, Issue 08, Pages: 170–175.

Paper Authors

K.RAJA RAO, N.G.N PRASAD

KAKINADAINSTITUTEOFENGINEERINGANDTECHNOLOGYFOR WOMEN,KORANGI,ANDHRAPRADESH,INDIA,533461

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

EFFICIENT COMPARISON-FREE SORTING ALGORITHM FOR HIGH EFFICIENCY IN O(N)

[1]K.RAJA RAO, [2]N.G.N PRASAD

[1]M.TECHVLES,DEPTOFE.C.E,KAKINADAINSTITUTEOFENGINEERINGAND TECHNOLOGY,KORANGI,ANDHRAPRADESH,INDIA,533461

[2]ASSOSIATE PROFESSOR, KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY,KORANGI,ANDHRAPRADESH,INDIA,533461

Abstract

This project proposes a hybrid approach for the enhancement of aerial images obtained from UAV/MAV cameras as well as an Application Specific Integrated Circuit (ASIC) design based sorting algorithm that excludes complex circuitry, but uses registers that hold the elements and their respective occurrences in the input set and employ matrix mapping to perform sorting. We propose a novel sorting algorithm that sorts input data integer elements on-the-fly without any comparison operations between the data—comparison-free sorting. We present a complete hardware structure, associated timing diagrams, and a formal mathematical proof, which show an overall sorting time, in terms of clock cycles, that is linearly proportional to the number of inputs, giving a speed complexity on the order of O(N). Our hardware-based sorting algorithm precludes the need for SRAM-based memory or complex circuitry, such as pipelining structures, but rather uses simple registers to hold the binary elements and the elements' associated number of occurrences in the input set, and uses matrix-mapping operations to perform the sorting process. Thus, the total transistor count complexity is on the order of O(N). We evaluate an application specified integrated circuit design of our sorting algorithm for a sample sorting of N = 1024 elements of size K = 10-bit using 90-nm Taiwan Semiconductor Manufacturing Company (TSMC) technology with a 1 V power supply.

Keywords: comparison free, Gigahertz clock cycle, one-hot weight representation, sorting algorithms, SRAM, speed complexity *O(N)*.

1. INTRODUCTION

The orders used are either in numerical order or lexicographical order. Sorting arranges the integer statistics into growing or decreasing order and an array of strings into alphabetical order. It can also be referred to as ordering the data. Sorting is taken into consideration as one of the maximum essential tasks in lots of pc packages for the motive that looking a sorted array or list takes less time when in comparison to an unordered or unsorted list. There had been many tries made to research the complexity of sorting algorithms and plenty of exciting and precise sorting algorithms have been proposed. There are extra benefits inside the observe of sorting

algorithms further to know-how the sorting techniques. These researches have gained a significant amount of electricity to remedy many other problems. Even although sorting is one of the extraordinarily studied problems in pc science, it stays the most preferred integrative algorithm trouble in practice. Moreover, every set of rules has its own benefits and disadvantages. For in-stance, bubble type might be efficient to type a small variety of items, . On the alternative hand, for a large number of objects brief type would perform thoroughly. There-fore, it isn't always thinkable that one sorting method is higher than another sorting technique. Moreover the performance of each sorting algorithm is predicated upon the statistics being sorted and the device used for sorting [1].

In popular, simple sorting algorithms perform two operations consisting of compare factors and assign one detail. These operations continue over and over until the statistics is taken care of [2]. Moreover, choosing a good sorting set of rules depending upon several factors along with the size of the enter statistics, to be had important memory, disk length, the volume to which the list is already taken care of and the distribution of values [1]. To measure the performance of different sorting algorithm we need to remember the subsequent statistics which includes the range of operations completed, the execution time and the distance required for the set of rules.

2. RELATED STUDY:

To address these challenges, much research has focused on architecting customized hardware designs for sorting algorithms in order to fully utilize the hardware resources and provide custom, cost-effective hardware processing. However, due to the inherent complexity of the sorting algorithms, efficient hardware implementation is challenging. To realize fast and power-efficient hardware sorting, a significant amount of hardware resources are required, including, but not limited to, comparators, memory elements, large global memories, and complex pipelining, in addition to complicated local and global control units. Most prior work on hardware sorting designs are implemented based on some modification of traditional mathematical algorithms, or are based on some modified network of switching structures with partially parallel computing processing and pipelining stages. In these sorting architectures, comparison units are essential components that are characterized by high-power consumption and feedback control logic delays. These sorting methods iteratively move data between comparison units and local memories, requiring wide, high-speed data buses, involving numerous shift, swap, comparison, and store/fetch operations, and have complicated control logic, all of which do not scale well and may need specialization for certain data-type particulars. Due to the inherent mixture of data processing and control logic within the sorting structures processing elements, designing these structures can be

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

cumbersome, imposing large design costs in terms of area, power, and processing time. Furthermore, these structures are not inherently scalable due to the complexity of integrating and combining the data path and control logic within the processing units, thus potentially requiring a full redesign for different data sizes, as well as complex connective wiring with high fan-out and fan-in in addition to coupling effects, thus circuit timing issues are challenging to address. Additionally, if multiple processors are used along with pipelining stages and global memories, the data must be globally merged from these stages to output the complete final sorted data set. To address these challenges, in this paper, we propose a new

4. METHODOLOGY

Leveraged a bitonic sorting network to more efficiently map the methodology considering energy and memory overheads for FPGA devices. Further advances of that work [48] presented novel and improved cost-performance tradeoffs, as well as identification of some Pareto optimal solutions trading off energy and memory overheads. Additional work [4] developed a framework that composes basic sorting architectures to generate a cost-efficient hybrid sorting architecture, which enabled fast hardware generation customized for heterogeneous FPGA/CPU systems. Even though all of these designs reported linear sorting delay times as the number of input elements increased, the authors did not include the initialization times for the required arrays/matrices, nor was the worst case sorting time evaluated. Furthermore, each design either required arrays to store the input elements, associated arrays for the rank operations and data routing, or had to globally merge the intermediate sorted array partitions. These array elements required a significant amount of local and global input–output data routing, SRAM-based memory, and control signals, where the local control logic communicated with each processing unit partition and the global control unit. This layout complicates adapting the design to different input data bit-widths. Additionally, since the control signals and data path wiring was intertwined, circuit design bugs were challenging to locate, in turn leading to high-cost design. This example operates as follows.

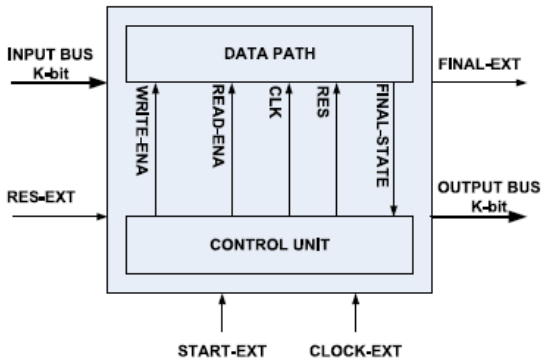

Fig.4.1. Block diagram of the hardware structure for our sorting algorithm.

The inputted elements are inserted into a binary matrix of size $N \times 1$, where each element is of size k-bit (in this example N = 4 and k = 2 bit). Concurrently, the inputted elements are converted to a one hot weight representation and stored into a one-hot matrix of size $N \times H$, where each stored element is of size H-bit and $H=N$giving a one-hot matrix of size N-bit $\times N$-bit. The one-hot matrix is transposed to a transpose

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

matrix of size $N \times N$, which is multiplied by the binary matrix—rather than using comparison operations—to produce the sorted matrix. For repeated elements in the input set, the one-hot transpose matrix stores multiple "1s" (equal to the number of occurances of the repeated element in the input set) in the element's associated row, where each "1" in the row maps to identical elements in the binary matrix, an advantage that will be exploited in the hardware design (Section V). For example, if the input set matrix is [2; 0; 2; 1], then the transpose matrix is [0 0 0 0; 1 0 1 0; 0 0 0 1; 0 1 0 0]. Notice that the second row contains two "1s," such that when the transpose matrix is multiplied by the second row in the binary matrix, both "1" occurances in the transpose matrix are mapped to the "2" in the binary matrix. Therefore, the multiply operation can be simply replaced with a mapping function using a tri-state buffer (Section V). Additionally, the first row in the transpose matrix has no element in the first position (i.e., element 3 is not in the binary matrix since 3 is not in the input set). The absence of this element can be recorded using a counting register for each inputted element (Section V), and this register records the number of occurences of this element in the binary matrix, which in this case would be "0" for element 3.

5. SIMULATION RESULTS

With respect to all evaluated results, our comparison-free sorting design provides an efficient linear scalability of O(*N*). Our design uses simple registers (flag, order, and sorted registers) that are accessed on both the rising and falling clock edges, and simple standard CMOS components with a forward flowing data movement architecture. Even though our design shows a linear performance cost of O(*N*), our hardware design is recommended for data element set sizes of less than 216 due to practical integration into large computing IC devices (e.g., graphics engines, routers, grid controllers.), where the sorting hardware accounts for no more than 10% of the IC's characteristics (power and area).

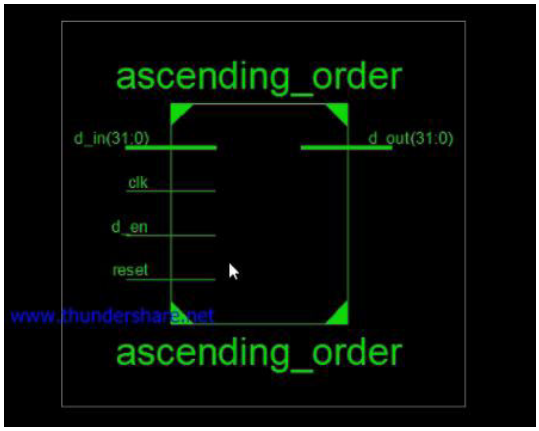

Fig.5.1. Model diagram.

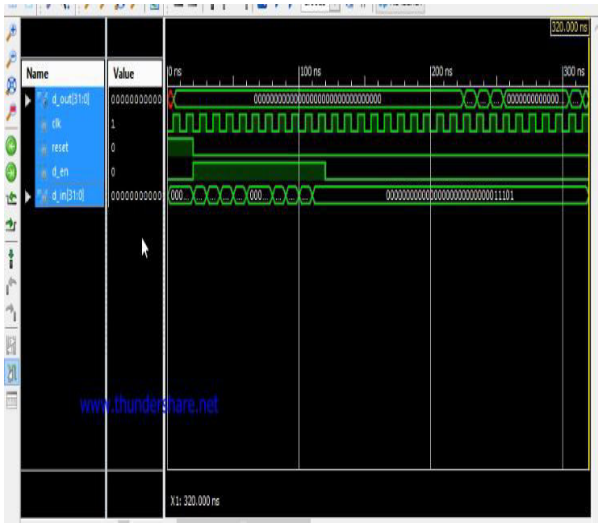

Fig.5.2. Simulation model 1.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Fig.5.3. Simulation model 2.

CONCLUSION

In this paper, we proposed a novel mathematical comparison-free sorting algorithm and associated hardware implementation. Our sorting design exhibits linear complexity O(*N)* with respect to the sorting speed, transistor count, and power consumption. This linear growth is with respect to the number of elements *N* for *N* = 2*K* where *K* is the bit width of the input data. The slope of the linear growth rate is small, with a growth rate of approximately 6 for the transistor count and power consumption, and 1.5 for the sorting speed. The order complexity and growth rates are due to simple basic circuit components that alleviate the need for SRAM-based memory and pipelining complexity. Our mathematically-simple algorithm streamlines the sorting operation in one forward flowing direction rather than using compare operations and frequent data movement between the storage and computational units, as with other sorting algorithms. Our design uses simple standard library components including registers, a one-hot decoder, a one detector, an incrementer/ decrementer, and a *PC*, combined with a simple control unit that contains a small amount of delay logic.

REFERENCES

[1] F.-C. Leu, Y.-T. Tsai, and C. Y. Tang, "An efficient external sorting algorithm," *Inf. Process. Lett.*, vol. 75, pp. 159–163, Sep. 2000.

[2] J. L. Bentley and R. Sedgewick, "Fast algorithms for sorting and searching strings," in *Proc. 8th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, Jan. 1997, pp. 360–369.

[3] L. Xiao, X. Zhang, and S. A. Kubricht, "Improving memory performance of sorting algorithms," *J. Experim. Algorithmic*, vol. 5, no. 3, pp. 1–20, 2000.

[4] P. Sareen, "Comparison of sorting algorithms (on the basis of average case)," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 3, pp. 522–532, Mar. 2013.

[5] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani, "AA-SORT: A new parallel sorting algorithm for multi-core SIMD processors," in *Proc. 16th Int. Conf. Parallel Archit. Compil. Techn. (PACT)*, 2007, pp. 189–198.

[6] V. Kundeti and S. Rajasekaran, "Efficient out-of-core sorting algorithms for the parallel disks model," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1427–1433, 2011.

[7] G. Capannini, F. Silvestri, and R. Baraglia, "Sorting on GPUs for large scale datasets: A thorough comparison," *Int. Process. Manage.*, vol. 48, no. 5, pp. 903–917, 2012.

[8] D. Cederman and P. Tsigas, "GPU-Quicksort: A practical quicksort algorithm

for graphics processors," *ACM J. Experim. Algorithmics (JEA)*, vol. 14, Dec. 2009, Art. no. 4.

[9] B. Jan, B. Montrucchio, C. Ragusa, F. G. Ghan, and O. Khan, "Fast parallel sorting algorithms on GPUs," *Int. J. Distrib. Parallel Syst.*, vol. 3, no. 6, pp. 107–118, Nov. 2012.

[10] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore GPUs," in *Proc. 23rd IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–10.