A Simple Architecture for Constant Time Sorting Machines

Tsong-Chih Hsu and Sheng-De Wang

Dept. of Electrical Engineering
EE Building, Rm. 441
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract- In this paper, we propose a constant time sorting algorithm on an array composed of comparators and single-pole-double-throw switches, which is far more feasible than other constant time sorting algorithms [21]-[23]. Our results shown that the algorithm uses time $T = \Theta(1)$ and area $A = O(N^3)$. This nearly matches the $AT^2 = \Omega(N^2 \log^2 N)$ lower bound for sorting in the VLSI model.

Keywords: constant time sorting, sorting by ranking, enumeration sort, parallel algorithm, mesh of trees.

1. Introduction

Many exciting developments have made in the field of computerized sorting since publications of the special issues on this subject 31 years ago in *Communications of the ACM* and 9 years ago in *IEEE Transactions on Computers*. These special issues cover several aspects of sorting from both theoretical and practical points of view. An early treatment of the subject of sorting networks is provided in [1]. The basic idea of enumeration sort is due to [2]-[3]. Networks for odd-even sort and bitonic sort were first described in Batcher's seminal paper [4]. Many researches extended Batcher's fundamental ideas and adapted them to a variety of parallel architectures. Such work is described, for examples, in [5]-[14]. Sequential sorting has been studied extensively for many years. Its best time complexity, $\Theta(N \log N)$, is well known. On some parallel computation models [15]-[16], the parallel counterpart of sequential sorting algorithms, have $O(\log N)$ complexity. There has also been much research in sorting algorithms for parallel processors [17-20].

Bilardi and Preparata [16] describe a VLSI implementation for sorting N numbers which uses time $T = O(\log N)$ and area $A = O(N^2)$. This matches the $AT^2 = \Omega(N^2 \log^2 N)$ lower bound for sorting in the VLSI model. Leighton [15] gives another solution to designing optimum $AT^2 = \Omega(N^2 \log^2 N)$ VLSI networks for sorting N numbers.

Constant time sorting can be achieved on an extremely powerful machine model, the Concurrent-Read Concurrent-Write Parallel Random Access Machine (CRCW PRAM), in which simultaneous accesses to the same memory location are allowed and the write conflict resolution process is to store the sum of all numbers that are written to the same memory location [21]. Although powerful, this machine is too idealistic to be implemented with the current hardware technology. Another constant time sorting algorithm is drived by [22], in which a three-dimensional processor array with a reconfigurable bus system is used. The processor array consists of N triangular arrays whose bottom processors are connected to form an $N \times N$ square array, where N is the number of data items to be sorted. Since this machine requires a three-dimensional array of $O(N^3)$ processors with a reconfigurable bus system, it is somewhat costly to be implemented with the current hardware technology. Recently, Chen and Chen [23] further developes a constant time sorting algorithm by using a 3-D reconfigurable mesh with only $O(N^{3/2})$ processors. Moreover, they further extend the result to k-dimensional reconfigurable meshes for $k \geq 3$. Their results are: an $O(4^{k+1})$ time sorting algorithm is obtained by using an $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$ k-dimensional reconfigurable mesh of size $O(N^{1+1/(k-1)})$. In this paper, we propose a constant time sorting algorithm based on an array of comparators and single-pole-double-throw switches.

The main differences between our algorithm and that of [23] are noted in the following. (1) The system of [23] is a general system, but we propose a special subsystem. (2) The architecture of [23] is a reconfigurable mesh, but we do not use it. (3) The processor element of [23] is a processor, but we just use comparators and single-pole-double-throw switches. (4) The constant factor of the complexity of [23] is large, but our algorithm is small. If we define the propagation delay of a comparator and the propagation delay of a single-pole-double-throw switch as τ_c and τ_s, respectively, we will show that the shortest time elapsed is $\tau_c + \tau_s$ for our algorithm; in general, the order of τ_c or τ_s is nsec. So, we say that the constant factor of our method is small, but the algorithm of [23] consists of six steps. The major operations in steps 1, 3, 5, and 6 are sorting r data items on an $r/m \times m \times r/m$ reconfigurable submesh. Operations in steps 2 and 4 are " Perform the transpose operation of the $r \times s$ matrix on the $r/m \times ms \times r/m$ 3-D reconfigurable mesh" and "Perform the inverse transpose operation of the $r \times s$ matrix on the $r/m \times ms \times r/m$ 3-D reconfigurable mesh", respectively. Although their result is an $O(4^{3+1})$ time sorting algorithm for $k = 3$, the constant factor is too large. By the way, since [23] is an $O(4^{k+1})$ time sorting algorithm, if k is too large, then it is may not mske sense.

Section II will describe the concept of sorting of ranking. The architecture for the sorting algorithm is presented in section III. Finally, the conclusion is given in section IV.

2. Sorting by Ranking

In this paper, we formally define the sorting problem:

Input: A sequence of N values $\langle a_0, a_1, \cdots, a_{N-1} \rangle$.

Output: A permutation $\langle s_0, s_1, \cdots, s_{N-1} \rangle$ of the input sequence such that $s_0 \leq s_1 \leq \cdots \leq s_{N-1}$.

Given an input sequence such as <50, 34.3, 99, -23>, a sorting algorithm should return as output the sequence <-23, 34.3, 50, 99>.

The rank R_i of a_i is defined to be the position of a_i in the sorted sequence minus one. For example, if $a_0 = 50$, $a_1 = 34.3$, $a_2 = 99$, $a_3 = -23$, then the sorted sequence is <-23, 34.3, 50, 99> and $R_0 = 2$, $R_1 = 1$, $R_2 = 3$, $R_3 = 0$. After the rank of each input data is determined, we know its position in the sorted sequence.

Applying the ranking concept to the sorting problem, the pay-off matrix of this sorting problem is shown in Table 1.

In Table 1, we define the pay-off values and the ranking parameters:

$$a_{ij} = \begin{cases} 1, \text{ if } a_i > a_j \\ 0, \text{ if } a_i \le a_j \end{cases} \qquad (1)$$
$$R_i = \sum_j a_{ij}$$

where $0 \le i, j \le N-1$. Then the sorted result can be represented in the output values s_0, s_1, ..., s_{N-1} with s_0 being the smallest value of $\forall R_i$, s_1 being the second value of $\forall R_i$, and so on. The result is similar to the enumeration sort which has been called by various, including the "orthogonal tree network" and the "mesh of trees".

In Table 1, **all a_{ij} can be constructed simultaneously**. There are many high speed differential comparators; for example the LM161/LM261/LM361 is a very high speed differential input, complementary TTL output voltage comparator. We can use them to construct the sorting circuits as shown in Figure 1. The basic module, a comparison module, for the digital-computation condition is very similar to Figure 1; it is easy to implement by hardware or simulate by software.

In the Figure 1, the time complexity is $\Theta(1)$, using $\frac{1}{2}N(N-1)$ comparators.

3. The architecture of sorting system

Figure 2 depicts the architecture of the proposed sorting system that can sort sequences of length N (where $0 \le i \le N-1$). In Figure 2, a_0, a_1, ..., a_{N-1} are the N input data. The sorted result is represented in the output values s_0, s_1, ..., s_{N-1} with s_0 being the smallest value and s_{N-1} being the largest value.

Table 1. A pay-off matrix.

	a_0	a_1	...	a_{N-1}	R_i
a_0	a_{00}	a_{01}	...	$a_{0(N-1)}$	R_0
a_1	a_{10}	a_{11}	...	$a_{1(N-1)}$	R_1
$\vdots$	...	...	...	...	$\vdots$
a_{N-1}	$a_{(N-1)0}$	$a_{(N-1)1}$	...	$a_{(N-1)(N-1)}$	R_{N-1}

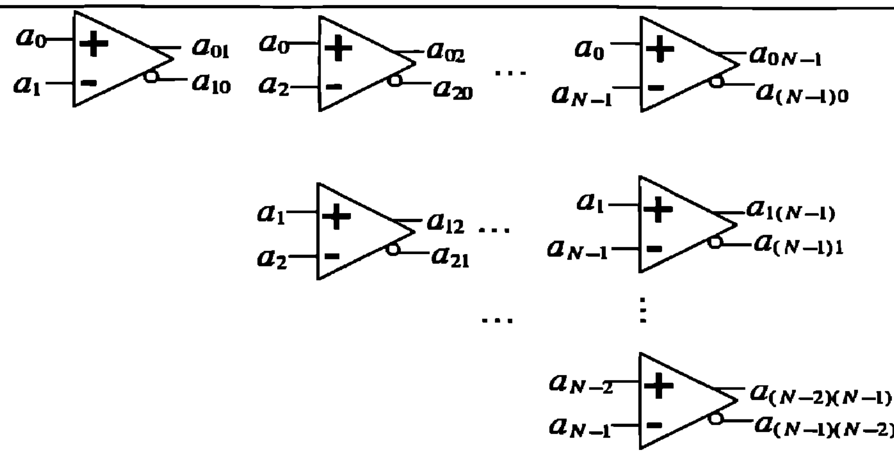

Figure 1. The a_{ij} generator.

In Figure 2, the detail of the a_{ij} generation is shown as Figure 1. As shown in Figure 2, we use the configuration $a_{i0}, a_{i1}, \ldots, a_{i(i-1)}, a_{i(i+1)}, \ldots, a_{i(N-1)}$ to control **simultaneously** the sorter consisting of $(N-1)$ stages of single-pole, double-throw switches. Specifically, the settings of all the switches of the ith sorting input element at the jth stages are congruent and are controlled by the binary variable $a_{i(j-1)}$. It is clear that if we feed a_i into the sorter and k a_{ij}'s in digits $\{a_{i0}, a_{i1}, \ldots, a_{i(i-1)}, a_{i(i+1)}, \ldots, a_{i(N-1)}\}$ are all equal to 1, a_i will emerge at the $(k+1)-$th terminal of the output, and sorting is completed in time proportional to $\Theta(1)$. For the digital-

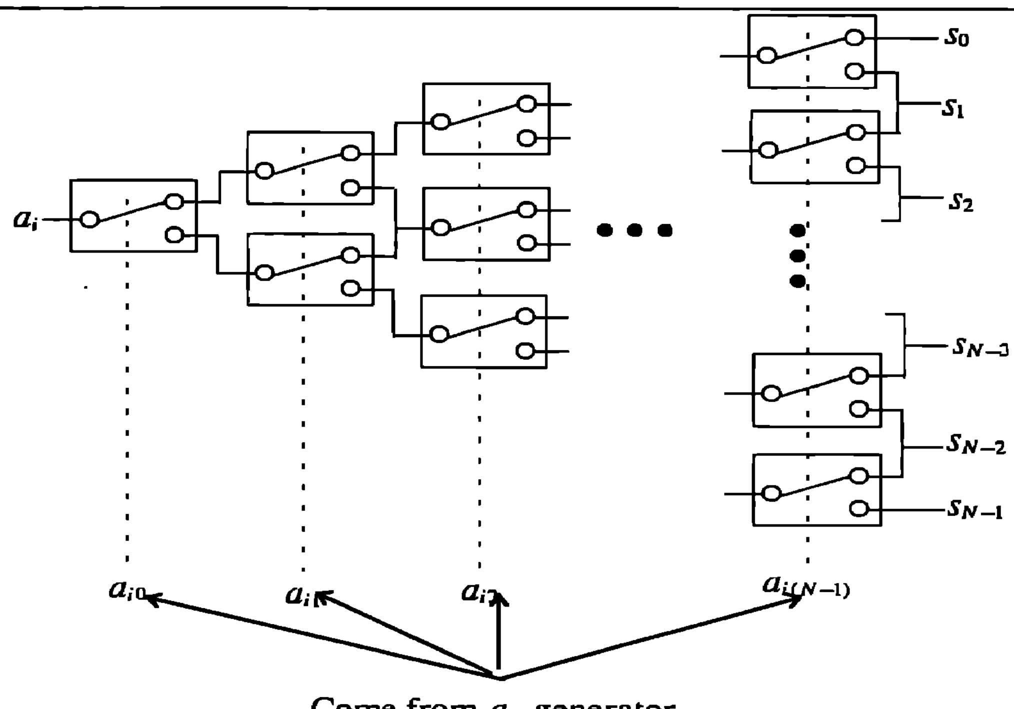

Figure 2. The architecture of sorter that can sort sequences of length N.

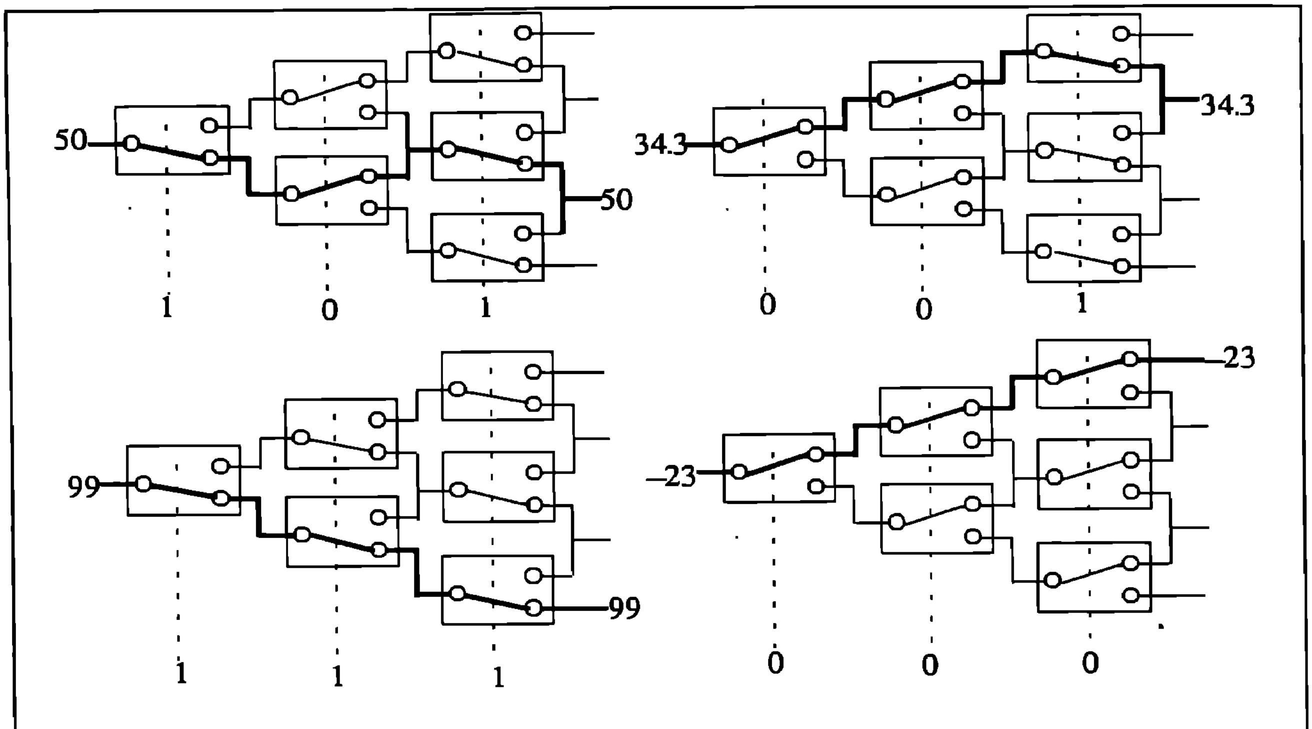

Figure 3. Controlling the switch units.

computation condition, we can apply a physical data moving or a logical data moving (link) technique to routing the input data. If we use the physical data moving, then the single-pole, double-throw switch in Figure 2 should be replaced by the single-pole, double-throw bus switch.

Example 3. If $a_0 = 50$, $a_1 = 34.3$, $a_2 = 99$, and $a_3 = -23$, then from (1), we have $R_0 = 2$, $R_1 = 1$, $R_2 = 3$, and $R_3 = 0$. From Figure 2, the final results are shown in Figure 3 and Figure 4: $s_0 = -23$, $s_1 = 34.3$, $s_2 = 50$, and $s_3 = 99$.

From Figures 1 and 2, the hardware

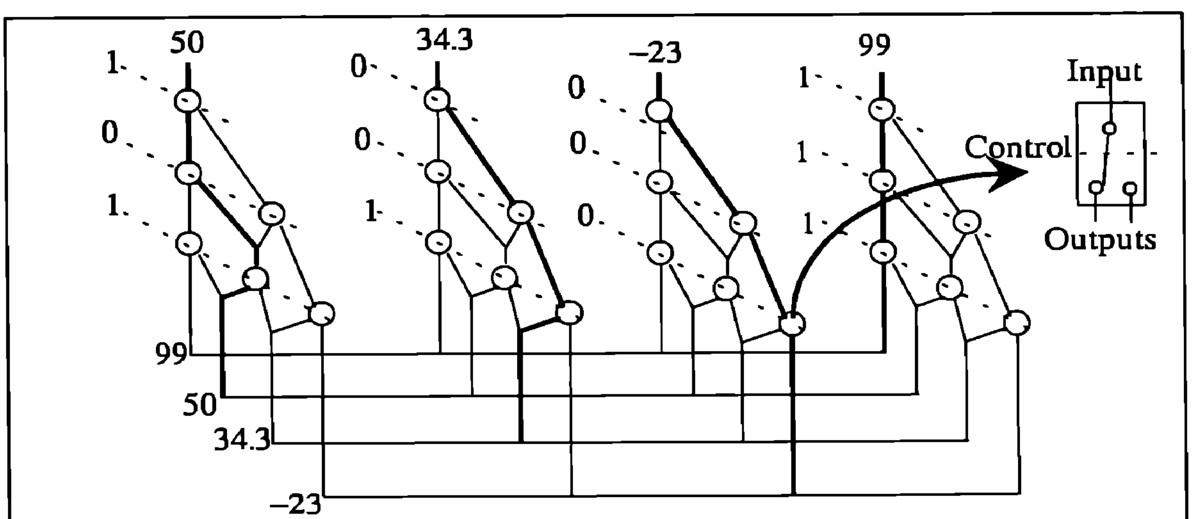

Figure 4. A four input data example.

complexity of the constant time sorter is noted in the following: (1) the computation of the digits $\{a_{ij}\}$ requires $\frac{1}{2}N(N-1)$ comparators; (2) each of the N switch control units contains

$$1+2+\cdots+N-1=\frac{1}{2}N(N-1)$$

switches.

Thus, we conclude that the constant time sorter requires a number of elements proportional to N^3. We use time $T=\Theta(1)$ and area $A=O(N^3)$. This nearly matches the $AT^2=\Omega(N^2\log^2 N)$ lower bound for sorting in the VLSI model.

Our final observation concerns sequences with repeated elements. The sorter as described above cannot handle such sequences. Indeed, if $a_i=a_j$, say, then $R_i=R_j$ and the two elements occupy the same position in the final sorted sequence. One way to solve this "collision" problem would be to assign a larger pay-off values to the element with the larger index. This can be accomplished by modifying the following test to (1):

$$a_{ij}=\begin{cases}1, \text{ if } (a_i>a_j) \text{ or } (a_i=a_j \text{ and } i>j)\\ 0, \quad\quad\quad\quad\quad\quad\quad\quad \text{otherwise}\end{cases}$$

In this way, the relative positions of equal elements are preserved in the sorted sequence.

4. Conclusion

In this paper, we proposed a very simple architecture for constant time sorting. This method is also suitable for VLSI implementation and has been analyzed using Thompson's model of VLSI. By using the parallel techniques, we have $\Theta(1)$ time complexity for the sorting problem. It could be a good choice for some special applications, for example, analog computations and high speed computations.

5. References

[1] D. E. Knuth, *The art of computer Programming, Vol. 3*, Addison-Wesley, reading, Massachusetts, 1973.

[2] D. E. Muller and F. P. Preparate, "Bounds to complexities of networks for sorting and for switching," *J. Assoc. Comput.*, Vol. 22, No. 2, pp. 195-201, 1975.

[3] D. Nath, S. N. Maheshwari, and P. C. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees," *IEEE Trans. Comput.*, Vol. c-32, No. 6, pp. 569-581, 1983.

[4] K. E. Batcher, "Sorting networks and their applications," *Proc. AFIPS 1968 Spring Joint Comput. Conf. Atlantic city, New Jersey*, pp. 307-314, 1968.

[5] H. S. Stone, "Sorting on STAR," *IEEE Trans. Software Engrg.*, Vol. se-4, No. 2 , pp. 138-146, 1978.

[6] H. S. Stone, "Parallel Processing with the perfect shuffle," *IEEE Trans. Comput., Vol. c-20, No. 2 , pp. 153-161, 1971.*

[7] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Comm. ACM, Vol. 20, No. 4, pp. 263-271, 1977.*

[8] D. Nassimi and S. Sahni, "Bitonic Sort on a mesh-computer parallel computer," *IEEE Trans. Comput.*, Vol. c-28, No. 1 , pp. 2-7, 1979.

[9] D. Nassimi and S. Sahni, "Parallel permutation and Sorting Algorithms and a new generalized connection network," *J. Assoc. Comput.*, Vol. 29, No. 3 , pp. 642-667, 1982.

[10] G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Trans. Comput., Vol. c-27, No. 1 , pp. 84-87, 1978.*

[11] F. P. Preparata, "New Parallel Sorting Schemes," *IEEE Trans. Comput.*, Vol. c-27, No. 7 , pp. 669-673, 1978.

[12] F. P. Prarata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Comm. ACM* Vol. 24, No. 5 , pp. 300-309, 1981.

[13] P. M. Flanders, "A unified approach to a class of data movements on an array processor," *IEEE Trans. Comput.*, Vol. c-31, No. 9 , pp. 809-819, 1982.

[14] M. Kumar and D. S. Hirschberg, "An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes," *IEEE Trans. Comput.*, Vol. c-32, No. 3 , pp. 254-264, 1983.

[15] F. T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, Vol. c-34, pp. 344-354, 1985.

[16] G. Bilardi and F. Preparata, "A minimum area VLSI network for O(log N) time sorting," *IEEE Trans. Comput.*, Vol. c-34, pp. 336-343, 1985.

[17] A. Borodin and J. E. Hopcroff, "Routing, merging and sorting on parallel models of computation," *in Proc. 14th Annu. ACM Symp. Theory Comput.*, pp. 338-344, 1982.

[18] C. P. Kruskal, "Searching, merging, and sorting in parallel computation," *IEEE Trans. Comput.*, Vol. c-32, pp. 942-946, 1983.

[19] L. Rudolph, "A robust sorting network," *IEEE Trans. Comput.*, Vol. c-34, pp. 326-335, 1985.

[20] D. Rotem, N. Santoro, and B. Sidneg, "Distributed Sorting," *IEEE Trans. Comput.*, Vol. c-34, pp. 372-375, 1985.

[21] S. G. Akl, *The design and analysis of parallel algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1989, pp. 93-96.

[22] B. F. Wang, G. H. Chen, and F. C. Lin, "Constant time sorting on a processor array with a reconfigurable bus system," *Information processing letters.*, Vol. 34, No. 4, pp. 187-192, 1990.

[23] Y. C. Chen and W. T. Chen, "Constant Time Sorting on Reconfigurable Meshes," *IEEE Trans. Comput.*, Vol. 43, No. 6, pp. 749-751, 1994.