Information Processing Letters 34 (1990) 187–192
North-Holland

24 April 1990

# CONSTANT TIME SORTING ON A PROCESSOR ARRAY WITH A RECONFIGURABLE BUS SYSTEM

Biing-Feng WANG, Gen-Huey CHEN and Ferng-Ching LIN

*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, People's Republic of China*

Communicated by S.G. Akl
Received 20 September 1989
Revised 21 December 1989

*Keywords*: Sorting, parallel algorithms, reconfigurable buses

## 1. Introduction

Sorting is undoubtedly one of the most fundamental problems in computer science. Many sequential sorting algorithms are available in the literatures [6] and it is well known that this problem requires $\Omega(n \log n)$ time in the worst case. To speed-up sorting, many parallel algorithms have been proposed on various parallel machines [2]. Constant time sorting can be achieved on an extremely powerful machine model, CRCW PRAM (concurrent-read concurrent-write parallel random access machine), in which simultaneous access (read or write) to the same memory location is allowed and the write conflict resolution process is to store the sum of all numbers that are written to the same memory location [3]. Although powerful, this machine is too idealistic to be implemented with the current hardware technology. On the other hand, networks of processors such as linear array, perfect shuffle, mesh, tree, and cube seem to be more feasible. However, the sorting times of these networks are bounded below by their diameters, which is by no means a constant.

Recently, many processor arrays have been supplemented with buses to decrease their diameters in order to enhance the system performance [1,4,8,15,20]. Buses can give processor arrays greater communication capabilities. They allow broadcasting and long-distance communication to be completed in constant time. A bus system whose configuration can be dynamically changed is called a *reconfigurable bus system*. Many reconfigurable bus systems such as bus automaton, reconfigurable mesh, and polymorphic-torus network have appeared in the literature [7,10,16,22].

A bus automaton [16,18] can be viewed as a cellular automaton with a locally switchable global communication network. By adjusting the local switches properly, straight, zig-zag, and staircase subbuses can be formed. Efficient algorithms on bus automata for many applications such as pattern recognition [9,17,19], language parsing [14], and string comparison [5], have been proposed. A reconfigurable mesh [10] consists of a square array of processors which are connected to a grid-shaped reconfigurable broadcast bus system. Within each processor, four locally controllable bus switches are built to adjust the configuration of the bus system. Some graph [11], image [12], and geometry problems [13] have been efficiently solved on the reconfigurable mesh. Like the reconfigurable mesh, a polymorphic-torus network [7] also consists of a square array of processors, but with a wrap around connection on each row and column. Efficient embeddings of tree, ring, mesh, pyramid, and hypercube can be realized by properly establishing the programmable local switches within each processor [7].

In this paper, we derive a constant time sorting

algorithm on a three-dimensional processor array equipped with a reconfigurable bus system, which is far more feasible than the CRCW PRAM model. The processor array consists of $N$ triangular arrays whose bottom processors are connected into an $N \times N$ square array, where $N$ is the number of data items to be sorted. The sorting algorithm is based on the well-known enumeration sort (also known as sorting by ranking) [6]. Data input, data comparisons, and data output are performed on the square array. The triangular arrays are responsible for ranking the data items during the sorting process.

We first introduce the execution of each triangular array in the next section. Then, we present the sorting algorithm in Section 3.

## 2. Summing a binary sequence on a triangular processor array

In this section we propose an $N \times N$ lower triangular processor array with six-neighbour connections to sum a binary sequence $b_0, b_1, \ldots, b_{N-1}$, where $b_i$, $0 \leqslant i \leqslant N-1$, is 0 or 1. Figure 1 shows such an array for $N = 6$. Each processor is identified by a unique index $(j,k)$. There are six ports, denoted by **U**, **D**, **N**, **S**, **UN**, and **DS**, built within each processor that connect to a reconfigurable broadcast bus. The configuration of the bus is dynamically changeable by adjusting a local switchable network in each processor. When two ports are connected by local switch, data that enter at one port leave the processor from the other port.

The summing algorithm consists of the following steps: (we explain the algorithm by summing the binary sequence 1, 1, 0, 1, 0, 1)

*Step* 0: Initially, $b_j$ is stored in processor $P_{j,0}$, for $0 \leqslant j \leqslant N-1$ (see Fig. 2(a)).

*Step* 1: Processor $P_{j,0}$, $1 \leqslant j \leqslant N-1$, sends a copy of $b_j$ to processor $P_{j-1,0}$.

*Step* 2: Every processor connects port **D** to port **U** to form vertically straight subbuses. Then, processor $P_{j,0}$, $0 \leqslant j \leqslant N-2$, broadcasts $b_j$ and $b_{j+1}$ on the corresponding subbus (see Fig. 2(b)).

*Step* 3: Processor $P_{j,k}$, $0 \leqslant j \leqslant N-2$ and $0 \leqslant k \leqslant N-1-j$, connects port **S** to port **N** if $(b_j, b_{j+1})$ is (0, 0), connects port **DS** to port **N** if $(b_j, b_{j+1})$ is (0, 1), connects port **S** to port **UN** if $(b_j, b_{j+1})$ is (1, 0), and connects port **DS** to port **UN** if $(b_j, b_{j+1})$ is (1, 1). The resulting configuration consists of several disjoint staircase subbuses. Then, the special processor $P_{N-1,0}$ broadcasts a signal "$*$" through port **UN** if $b_{N-1} = 1$ and through port **N** otherwise (see Fig. 2(c)).

*Step* 4: Processor $P_{0,k}$, $0 \leqslant k \leqslant N-1$, connects port **U** to port **D** to form a vertically straight subbus. Then, the processor $P_{0,k'}$, that received the signal "$*$" broadcasts the value $k'$ on the established subbus (see Fig. 2(d)).

*Step* 5: Processor $P_{0,0}$ computes the sum as $k' + b_0$ (see Fig. 2(e)).

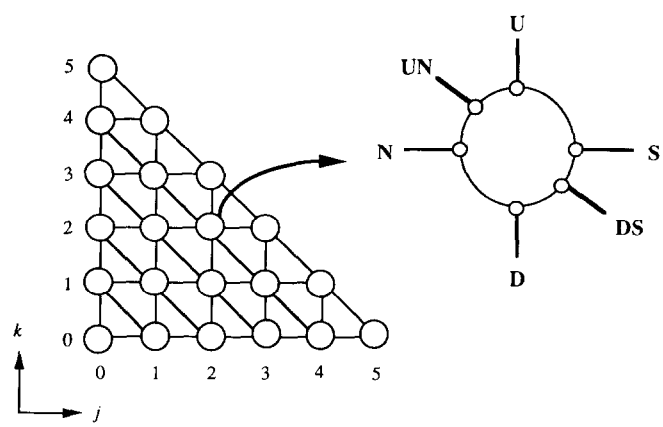



Fig. 1. A triangular processor array with a reconfigurable bus system.

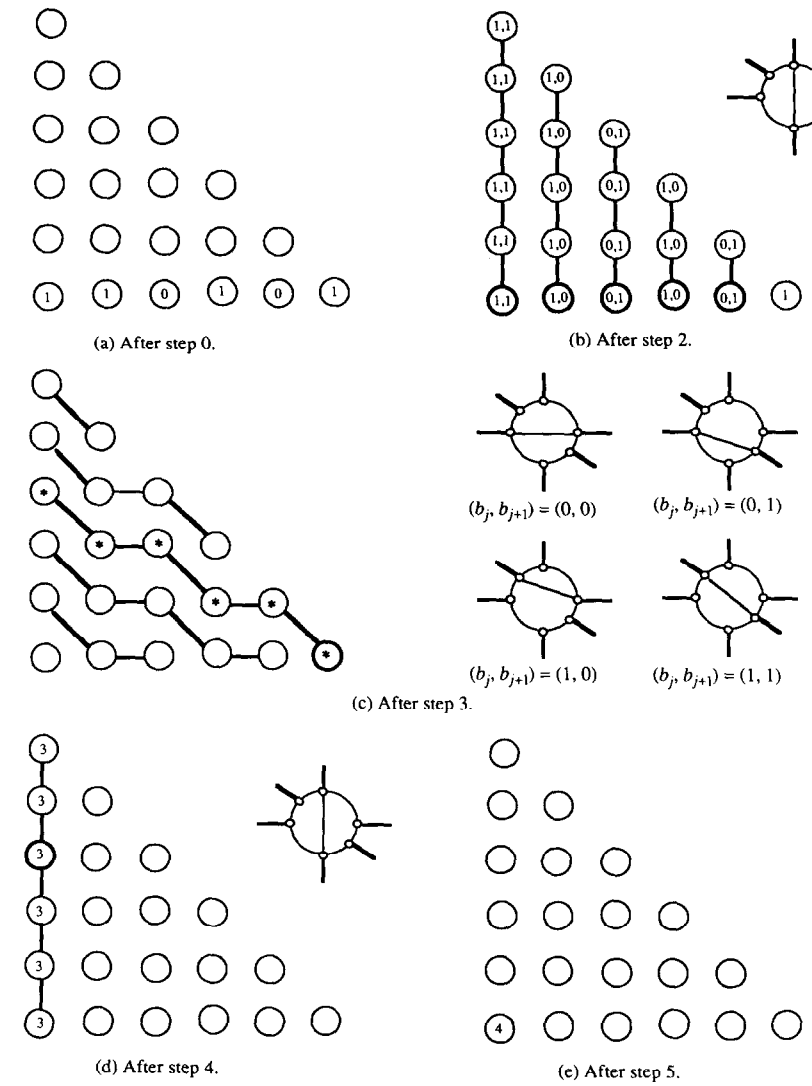



Fig. 2. Summing the binary sequence 1,1,0,1,0,1.

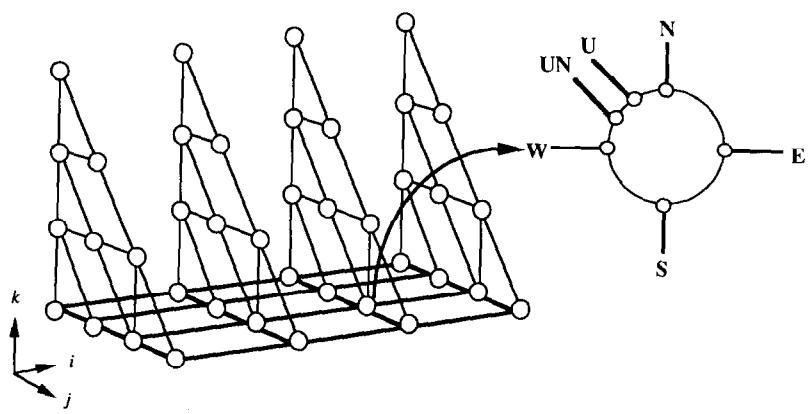
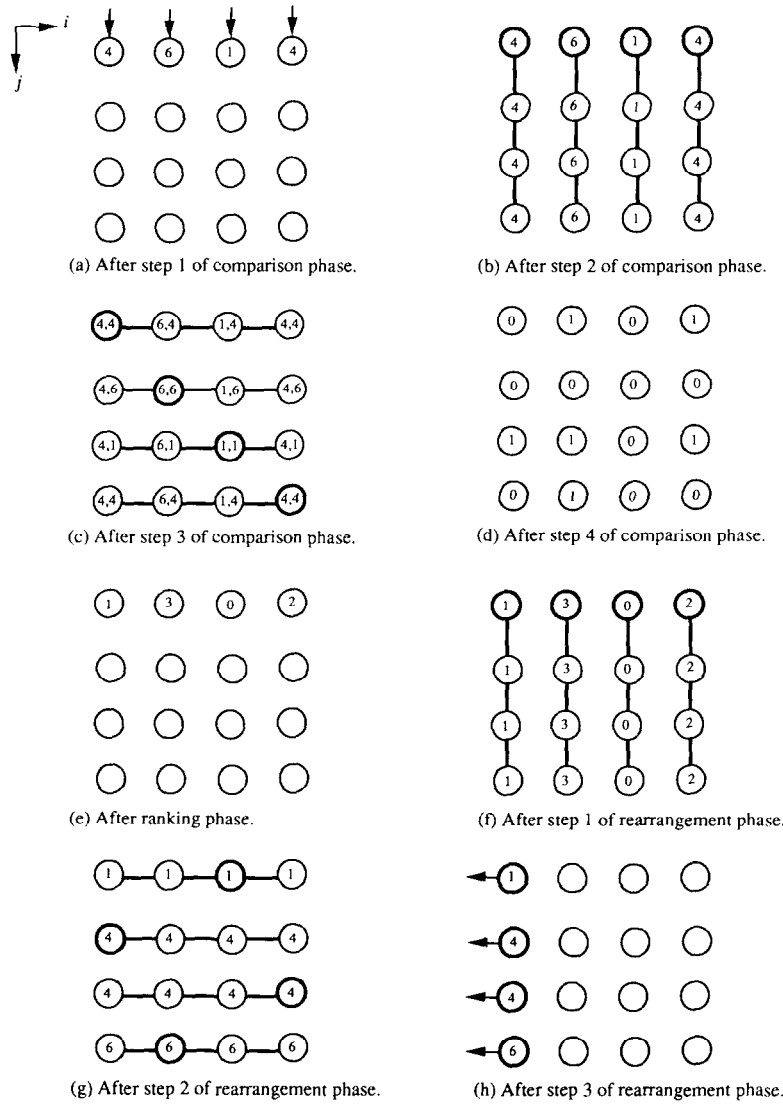



Fig. 3. Four triangular processor arrays whose bottom processors are connected into a square processor array.

It is clear that the algorithm takes only constant time. The correctness of the algorithm is assured by the following lemma.

**Lemma 2.1.** *The triangular processor array does compute the sum of the input binary sequence.*

**Proof.** Step 3 is the most crucial. In Step 3, the signal "$*$" enters a processor $P_{j,k}$ at port **S** as $b_{j+1} = 0$ and at port **DS** as $b_{j+1} = 1$, and leaves $P_{j,k}$ from port **N** as $b_j = 0$ and from port **UN** as $b_j = 1$. So, the signal goes up one stair step if a "1" is encountered and goes horizontally otherwise. After Step 3, in each column only one processor can receive the signal. If a processor $P_{j,k}$, $0 \leqslant j \leqslant N-2$, receives the signal, $k' = b_{j+1} + b_{j+2} + \cdots + b_{N-1}$. The desired sum $b_0 + b_1 + \cdots + b_{N-1}$ will be obtained in $P_{0,0}$ after Step 5. □




(a) After step 1 of comparison phase.

(b) After step 2 of comparison phase.

(c) After step 3 of comparison phase.

(d) After step 4 of comparison phase.

(e) After ranking phase.

(f) After step 1 of rearrangement phase.

(g) After step 2 of rearrangement phase.

(h) After step 3 of rearrangement phase.

Fig. 4. Sorting data items 4,6,1,4.

## 3. Sorting on a processor array with a reconfigurable bus system

The processor array consists of $N$ triangular arrays introduced in the last section. The bottom processors of the triangular arrays are connected into an $N \times N$ square array, as depicted in Fig. 3 for $N = 4$. Six ports **N**, **E**, **W**, **S**, **UN**, and **U** are built within each bottom processor. Each processor in the processor array is identified by a unique index $(i, j, k)$ and is connected to a reconfigurable broadcast bus through the ports.

The sorting algorithm is composed of three phases: comparison, ranking, and rearrangement. In the comparison phase, the data items are compared with each other. The ranking phase determines the rank of each data item using the results of the comparison phase. The rearrangement phase rearranges the data items into a nondecreasing sequence according to their ranks. Let $d_0, d_1, \ldots, d_{N-1}$ (not necessarily distinct) be the data items to be sorted. The execution of each phase is as follows: (we explain the algorithm by sorting the data items 4, 6, 1, 4)

**Comparison phase.**

*Step* 1: For $0 \leqslant i \leqslant N-1$, input data item $d_i$ into processor $P_{i,0,0}$ in parallel (see Fig. 4(a)).

*Step* 2: All processors in the square array connect port **N** to port **S** to form straight subbuses. Then, processor $P_{i,0,0}$, $0 \leqslant i \leqslant N-1$, broadcasts $d_i$ on the corresponding subbus (see Fig. 4(b)).

*Step* 3: All processors in the square array connect port **E** to port **W** to form straight subbuses. Then, processor $P_{j,j,0,0} \leqslant j \leqslant N-1$, broadcasts $d_j$ on the corresponding subbus (see Fig. 4(c)).

*Step* 4: Processor $P_{i,j,0}$, $0 \leqslant i \leqslant N-1$ and $0 \leqslant j \leqslant N-1$, sets $b_{i,j} = 1$ if $(d_i > d_j)$ or $((d_i = d_j)$ and $(i > j))$, and $b_{i,j} = 0$ otherwise (see Fig. 4(d)).

**Ranking phase.** As we have described in the last section, the triangular arrays can compute $r_i = b_{i,0} + b_{i,1} + \cdots + b_{i,N-1}$ in parallel, for $0 \leqslant i \leqslant N-1$. After the ranking phase, the rank $r_i$ of the data item $d_i$ is kept in processor $P_{i,0,0}$ (see Fig. 4(e)).

**Rearrangement phase.**

*Step* 1: All processors in the square array connect port **N** to port **S** to form straight subbuses. Then, processor $P_{i,0,0}$, $0 \leqslant i \leqslant N-1$, broadcasts $r_i$ on the corresponding subbus (see Fig. 4(f)).

*Step* 2: All processors in the square array connect port **E** to port **W** to form straight subbuses. Then, processors $P_{i,j',0}$, $0 \leqslant i \leqslant N-1$ and $j' = r_i$, broadcasts $d_i$ on the corresponding subbus (see Fig. 4(g)).

*Step* 3: Processor $P_{0,j,0}$, $0 \leqslant j \leqslant N-1$, outputs the data item that it has received in Step 2 (see Fig. 4(h)).

**Theorem 3.1.** *The three-dimensional array of* $O(N^3)$ *processors with a reconfigurable bus system can sort* $N$ *data items in constant time.*

The proof of Theorem 3.1 is immediate from Lemma 2.1 and the description of the sorting algorithm.

## 4. Concluding remarks

In this paper we have presented a constant time sorting algorithm on a three-dimensional processor array with a reconfigurable bus system. In fact, by embedding the triangular arrays into the square array, the dimension of the processor array can be reduced from three to two without increasing the processor number. The authors are currently trying to reduce the processor number from $O(N^3)$ to $O(N^2 \log^2 N)$, which is the hardware lower bound of sorting while retaining the constant time [21].

## References

[1] A. Aggarwal, Optimal bounds for finding maximum on array of processors with $k$ global buses, *IEEE Trans. Comput.* **35** (1) (1986) 62–64.

[2] S.G. Akl, *Parallel Sorting Algorithms* (Academic Press, Orlando, FL, 1985).

[3] S.G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice-Hall, Englewood Cliffs, NJ, 1989) 93–96.

[4] S.H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. Comput.* **33** (2) (1984) 133–139.

[5] D.M. Champion and J. Rothstein, Immediate parallel solution of the longest common subsequence problem, in: *Proc. International Conference on Parallel Processing* (1987) 70–77.

[6] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).

[7] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Trans. Comput.* **38** (9) (1989) 1345–1351.

[8] P. McKinley, Multicast routing in spanning bus hypercubes, in: *Proc. International Conference on Parallel Processing* 2 (1988) 204–211.

[9] J.R. Melby, Recognition of straight lines by bus automata using parallel processing, Ph.D. Thesis, The Ohio State University, Columbus, OH (1980).

[10] R. Miller, V.K. Prasanna Kumar, D. Reisis and Q.F. Stout, Meshes with reconfigurable buses, in: *Proc. 5th MIT Conference on Advanced Research in VLSI* (1988) 163–178.

[11] R. Miller, V.K. Prasanna Kumar, D. Reisis and Q.F. Stout, Data movement operations and applications on reconfigurable VLSI arrays, in: *Proc. International Conference on Parallel Processing* 1 (1988) 205–208.

[12] R. Miller, V.K. Prasanna Kumar, D. Reisis and Q.F. Stout, Image computations on reconfigurable VLSI arrays, in: *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognition* (1988) 925–930.

[13] R. Miller and Q.F. Stout, Efficient parallel convex hull algorithms, *IEEE Trans. Comput.* **37** (12) (1988) 1605–1618.

[14] J.M. Moshell and J. Rothstein, Bus automata and immediate languages, *Inform. and Control* **40** (1979) 88–121.

[15] V.K. Prasanna Kumar and C.S. Raghavendra, Array processor with multiple broadcasting, *J. Parallel and Distributed Comput.* **2** (1987) 173–190.

[16] J. Rothstein, On the ultimate limitations of parallel processing, in: *Proc. International Conference on Parallel Processing* (1976) 206–212.

[17] J. Rothstein, Toward pattern-recognizing visual prostheses, in: *Proc. IFAC Symposium, Control Aspects of Prosthetics and Orthotics*, Columbus (1982) 87–89.

[18] J. Rothstein, Bus automata, brains, and mental models, *IEEE Trans. Systems Man Cybernet.* **18** (4) (1988) 522–531.

[19] J. Rothstein and A. Davis, Parallel recognition of parabolic and conic patterns by bus automata, in: *Proc. International Conference on Parallel Processing* (1979) 288–297.

[20] Q.F. Stout, Mesh connected computers with broadcasting, *IEEE Trans. Comput.* **32** (9) (1983) 826–830.

[21] C.D. Thompson, The VLSI complexity of sorting, *IEEE Trans. Comput.* **32** (12) (1983) 1171–1184.

[22] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash and D.B. Shu, The image understanding architecture, COINS Tech. Rept. 87-76, University of Massachusetts, Amherst, MA (1987).