Adaptive Binary Sorting Schemes and Associated Interconnection Networks

Minze V. Chien, *Member, IEEE*, and A. Yavuz Oruç, *Senior Member, IEEE*

***Abstract*—Many routing problems in parallel processing, such as concentration and permutation problems, can be cast as sorting problems. In this paper, we consider the problem of sorting on a new model, called an adaptive sorting network. We show that any sequence of n bits can be sorted on this model in $O(\lg^2 n)$ bit-level delay using $O(n)$ constant fanin gates. This improves the cost complexity of Batcher's binary sorters by a factor of $O(\lg^2 n)$ while matching their sorting time. The only other network that can sort binary sequences in $O(n)$ cost is the network version of columnsort algorithm, but this requires excessive pipelining. In addition, using binary sorters, we construct permutation networks with $O(n \lg n)$ bit-level cost and $O(\lg^3 n)$ bit-level delay. These results provide the asymptotically least-cost practical concentrators and permutation networks to date. We note, of course, that the well-known AKS sorting network has $O(\lg n)$ sorting time and $O(n \lg n)$ cost, but the constants hidden in these complexities are so large that our complexities outperform those of the AKS sorting network until n becomes extremely large.**

***Index Terms*— Adaptive sorting, binary sorter, concentrator, permutation network, sorting network**

I. Introduction[1]

SORTING networks have received much interest because of their widespread use in many computations and algorithms. See [12] for an in-depth treatment of sorting networks, and [19], [23], for recent surveys. These networks are constructed by cascading constant fanin comparator switches, and are termed *nonadaptive* in that the comparisons at the switches are not predicated on any conditions [25]. Inputs upon entering a sorting network are compared and exchanged at the comparators, depending on their relative values, as shown in Fig. 1. Two parameters that are most commonly used to assess the performance of a sorting network are its *cost* and *depth*. The cost of a sorting network is the number of constant fanin comparator switches that it contains, and its depth is the maximum number of such switches on a path from an input to an output. The cost and depth of the network in Fig. 1 are 5 and 3, respectively.

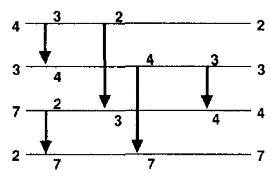


Fig. 1. A four-input sorting network.

Many models of sorting exist, but sorting networks seem to have been most resourceful among these in that many parallel sorting algorithms are adaptations of Batcher's odd-even merge and bitonic sorting networks [3], [12] and the more recent AKS sorting network [1], [19], [20]. In particular, the AKS sorting network construction settled a long-standing question about the existence of a sorting network with $O(n \lg n)$ cost and $O(\lg n)$ depth, and it had strong implications for the complexity of sorting on parallel computer models [14]. Despite its theoretical significance, however, the AKS sorting network is far from having any practical value, because of the large constants in its cost and depth complexities, and the problem of constructing an $O(n \lg n)$ cost and $O(\lg n)$ depth sorting network with small constants remains a challenging open problem.

In this paper, we examine the possibility of constructing sorting networks whose cost and depth complexities approach these expressions as best as possible. To facilitate this, we consider a new sorting network model, called an *adaptive sorting network*. The main difference between this model and a nonadaptive sorting network is that the latter is constructed by using only comparators, whereas an adaptive sorting network may have other components to check on conditions for comparing and routing its inputs through.[2] Our main problem in this model is to sort an arbitrary binary sequence of n elements. Apart from being significant in and of itself, sorting binary sequences plays quite a central role in concentration, permutation, and sorting problems. In fact, the concentration problem is equivalent to sorting binary sequences [6], and the permutation and sorting problems can be broken into a sequence of sorting steps on binary sequences.

The well-known zero-one principle dictates that any nonadaptive network of comparators that sorts an arbitrary binary sequence also sorts any "totally ordered" set of elements [12]. As such, a nonadaptive binary sorting network, i.e., one that sorts all binary sequences of length n, with $O(n \lg n)$ bit-

Manuscript received February 18, 1992; revised September 23, 1992. This work was supported in part by the National Science Foundation under Grant CCR-8708864, in part by the Minta Martin Fund of the School of Engineering at the University of Maryland, and in part by the National Aeronautics and Space Administration under Grants NAG-5-203 and NAG-5-914.

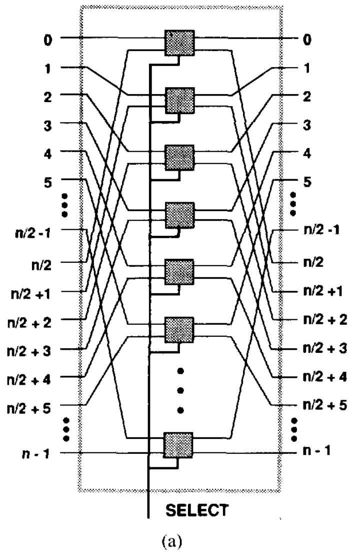
A. Yavuz Oruç is with the Department of Electrical Engineering and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA; e-mail: yavuz@eng.umd.edu.

Minze V. Chien is with PRC, Inc., McLean, VA, USA; e-mail: chien-minze@prc.com.

IEEE Log Number 9401200.

[1] All logarithms in this paper are in base 2, and $\lg n$ denotes $\lg_2 n$.

[2] A formal definition of an adaptive sorting network model is given in the next section.



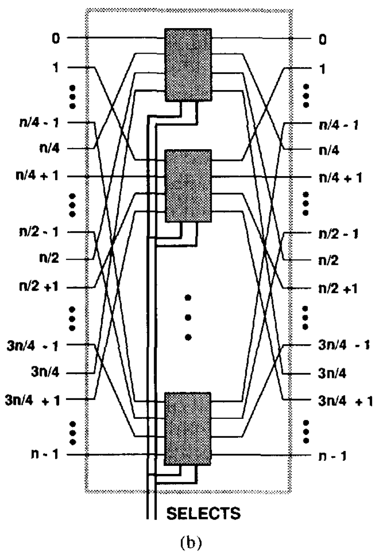


Fig. 2. Swapping networks. (a) Two-way swapper. (b) Four-way swapper.

level cost and $O(\lg n)$ bit-level depth (hereafter called cost and depth), where the constants in the order expressions are small, will have strong implications for sorting in general; but this seems highly unlikely, given the countless unsuccessful attempts on the problem.[3]

As an alternative, we explore in this paper the possibility of constructing adaptive networks to sort binary sequences within optimal bounds. We present three such networks:

1) An adaptive binary sorting network with $O(n \lg n)$ cost and $O(\lg n)$ depth using a prefix adder scheme,
2) An adaptive binary sorting network with $O(n \lg n)$ cost and $O(\lg^2 n)$ depth using a multiplexed merging scheme, and
3) An adaptive time-multiplexed binary sorting network with $O(n)$ cost, $O(\lg^2 n)$ depth, and $O(\lg^3 n)$ sorting time without pipelining or $O(\lg^2 n)$ sorting time with pipelining, all in bit level, using a time-multiplexed sorting scheme.

These network constructions compete well with the binary versions of the best-known sorting networks. The first two have $O(n \lg n)$ cost that is matched only by the AKS binary sorting network, but with an impractically large constant. The third construction has $O(n)$ cost, which is as good as the time-multiplexed network version of columnsort [14], in which the sorting steps are implemented by $n/\lg^2 n$-input Batcher's sorters and inputs are pipelined. We must note, however, that though the columnsort network requires inputs to each of its sorters being separately pipelined, our last network construction needs to pipeline the inputs through a single $n/\lg n$-input sorter. In addition to their low costs, the depth and sorting time complexities of our binary sorting networks match those of Batcher's binary sorting networks and binary columnsort networks.

We also note that there exist $O(n)$ bit-level cost and $O(\lg n)$ bit-level depth n-input Boolean sorting circuits, as reported in [17], [26]. These circuits cannot carry, or move the inputs through, however; they generate only sorted bits at their outputs. Therefore, they are outside the focus of this paper.

The rest of this paper is organized as follows. Section II describes our adaptive sorting network models. Section III presents our binary sorting networks. Section IV describes how binary sorters can be used to obtain concentrators and permutation networks. The paper is concluded in Section V.

II. Adaptive Sorting Models

In this section, we describe the two models that we use to construct our networks. In tallying the costs and depths of our networks, it will be assumed that each of 2×2 switch, 2×1 mulitiplexer, and 1×2 demultiplexer has unit cost and unit depth. First, we sketch the building blocks needed in these models.

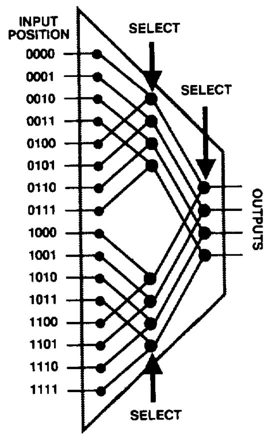
A. Two-Way Swapper

A two-way swapping network, or two-way swapper, as shown in Fig. 2(a), swaps two halves of its inputs before mapping them to its outputs, when its control signal is enabled. An n-input two-way swapper consists of a two-way shuffle connection, a single stage of $n/2$ 2×2 switches, a reversed two-way shuffle connection, and a single control signal. Its cost is $n/2$ and its depth is 1.

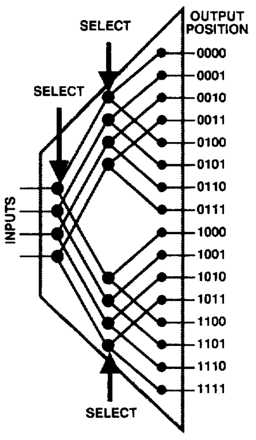
B. Four-Way Swapper

A four-way swapping network, or four-way swapper, as shown in Fig. 2(b), swaps four quarters of its inputs before mapping them to its outputs in any one of four ways, when

[3]Here "bit-level cost" refers to the number of constant fanin logic gates, and "bit-level depth" is the maximum number of such gates on a path from an input to an output.



(a)



(b)

Fig. 3. (a) (16, 4)-multiplexer. (b) (4, 16)-demultiplexer.

its select signals are set accordingly. These four swapping patterns are not fixed and vary with the use of a four-way swapper. Let 1, 2, 3, 4 denote the first, second, third, and fourth quarters of inputs (outputs) of the swapper. In particular, we will subsequently need two types of four-way swappers to affect the following two sets of permutations expressed in cycle notation: $\{(1)(23)(4), (1)(234), (13)(24), (134)(2)\}$, $\{(1)(2)(3)(4), (1)(243), (13)(24)\}$. These two four-way swappers will be referred to as IN-SWAP and OUT-SWAP networks, respectively.

An n-input four-way swapper consists of a four-way shuffle connection, a single stage of $n/4$ 4×4 switches, a reversed four-way shuffle connection, and two select signals. Its cost[4] is n, and its depth is 1.

C. Multiplexers

An $(m, 1)$-multiplexer is a device that can select and connect one of its m inputs to its only output. It can be realized by a balanced binary tree of lg m levels of (2, 1)-multiplexers. An (n, k)-multiplexer is a device that can select and connect any one of n/k groups of inputs, where each group consists of k inputs, to its k outputs according to the value of its lg (n/k)-bit select inputs, where $k \leq n$. An (n, k)-multiplexer can be formed by coupling k $(n/k, 1)$-multiplexers. A $(16, 4)$-multiplexer is shown in Fig. 3(a). The four groups of inputs are identifed by the leftmost two bits of the binary codes assigned to the inputs; those having the same two leftmost bits are in the same group. The desired group of inputs can be selected by activating the $(2, 1)$-multiplexers with these group identifier or select bits. Assuming that it is formed by coupling k $(n/k, 1)$-multiplexers, an (n, k)-multiplexer exacts n costs and lg (n/k) depth.

D. Demultiplexers

A $(1, m)$-demultiplexer is a device that can connect its only input to one of its m outputs. A $(1, m)$-demultiplexer can be constructed by a balanced binary tree with lg m levels of $(1, 2)$-demultiplexers. A (k, n)-demultiplexer is a device that can connect k inputs to any one of its n/k groups of outputs according to the value of its lg (n/k)-bit select inputs, where $k \leq n$. A (k, n)-demultiplexer can be formed by coupling $k\,(1, n/k)$-demultiplexers, in which case its cost is n, and its depth is lg (n/k). A $(4, 16)$-demultiplexer is shown in Fig. 3(b). We use these four building blocks to form our two adaptive sorting network models. More precisely, we use the following models.

Network Model A: Combinational Adaptive Sorter: The adaptive sorting networks in this model are constructed from two-way and four-way swappers, and other combinational circuits. Thus, all of our constructions under this model are combinational circuits encompassing constant fanin logic gates.

Network Model B: Time-Multiplexed Adaptive Sorter: In this model, we use all four building blocks and assume that there is a global clock that times our steps for moving various groups of inputs through (n, k)-multiplexer and (k, n)-demultiplexer blocks. To improve the time bounds without increasing the cost bounds, we also assume that inputs can be pipelined in this model. The adaptive sorting networks under this model can be viewed as simple sequential or clocked circuits.

III. Adaptive Binary Sorting Networks

We present three adaptive binary sorting networks, the first two are based on the network model A, and the third belongs to the network model B. For ease of discussion, but with no loss of generality, we shall assume that all of our sorting networks use power of 2 inputs.

A. Network 1 (Prefix Binary Sorter)

The first binary sorting network is based on a variant of odd-even merge sorting network [3]. The construction of this sorting network for 16 inputs is shown in Fig. 4(b), and can easily be extended to larger numbers of inputs. This sorter construction is similar to Batcher's odd-even merge sorting

[4]This cost is normalized to the number of 2×2 switches where the cost of each 4×4 switch is roughly equivalent to the cost of four 2×2 switches.

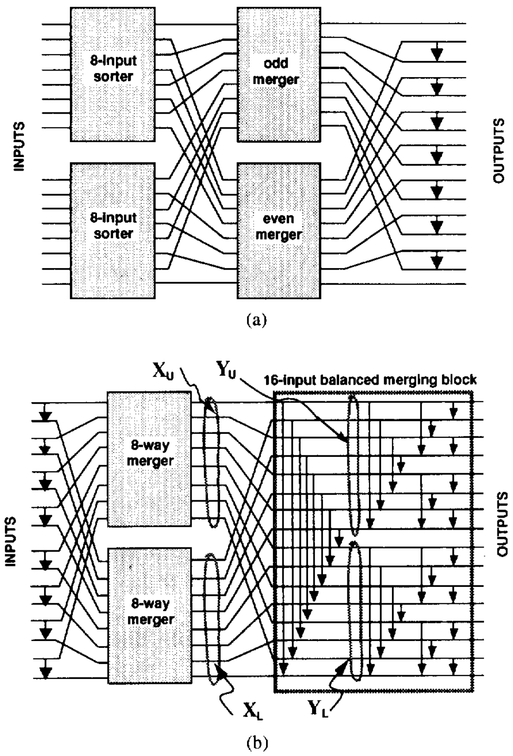


Fig. 4. 16-input odd-even merge sorting networks. (a) Batcher's odd-even merge sorting network. (b) An alternative odd-even merge sorting network. The stage of comparators on the left and the shuffle connection are redundant. They are shown to emphasize the relation between a two-way odd-even merge sorting network and an $n/2$-way odd-even merge sorting network.

network (Fig. 4(a)), except that the two $n/2$-input sorters in Batcher's network are replaced by $n/2$ two-input sorters, the even and odd $n/2$-input mergers are replaced by $n/2$-way mergers, and the merging of sorted subsequences is done by a network called a *balanced merging block* [8], [9], [24]. In other words, the outputs of the binary comparators in the first stage contain $n/2$ sorted sequences, each with two elements, and the rest of the network merges them by using an odd-even merge scheme. The $n/2$-way mergers are actually $n/2$-input sorters, where each of the $n/2$ sorted sequences to be merged contains a single element. (In fact, a moment's reflection reveals that merging sets, each containing only one element, amounts to sorting them.) As stated in Fig. 4(b), the first stage of comparators and the shuffle connection are redundant and are included only to demonstrate the extension of Batcher's odd-even merge sorting networks.

Obviously, the balanced merging block that follows the two $n/2$-way mergers in the network of Fig. 4(b) is more complex than $n/2-1$ two-input comparators used in Batcher's odd-even merge sorter. The trade-off is that the sorting problem on the input side in this case is much simpler; we need only $n/2$ two-input sorters (i.e., two-input comparators) rather than two $n/2$-input sorters. It is left to the reader to examine this trade-off between the sorting and merging steps by considering other distributions of the overall sorting problem between the two steps.

For binary sequences, the balanced merging block used on the right side of Fig. 4(b) has $O(n \lg n)$ cost and $O(\lg n)$ depth. If the two $n/2$-way merging networks are recursively replaced by half-size odd-even merge sorting networks, then a binary sorting network with $O(n \lg^2 n)$ cost and $O(\lg^2 n)$ depth is obtained.

This odd-even merge sorting scheme works for arbitrary numbers. For binary inputs (1-bit inputs), the cost of the balanced merging block can be reduced from $O(n \lg n)$ to $O(n)$ by observing the following facts.

Definition 1: Let $\mathcal{A}_n$ be the set of all binary sequences of length n starting with any multiple of 00 or any multiple of 11, followed by any multiple of 01 or any multiple of 10, and that followed by any multiple of 00 or any multiple of 11. This is stated more precisely by the following regular expression:

$$\mathcal{A}_n = \{0,1\}^n \cap [((00)^* + (11)^*)((01)^* + (10)^*)((00)^* + (11)^*)].$$

As an example, 0000/1010, 00/1010/11, 101010/11, 00/0101/11, 11111111 are all elements of $\mathcal{A}_8$.||

Remark: Note that zero multiples of 00, 01, 10, and 11 are allowed. Note also that any sorted binary sequence of length n belongs to $\mathcal{A}_n$.||

Theorem 1: Let X_U and X_L be any two ascendingly sorted binary sequences of length $n/2$. If X_U and X_L are concatenated and shuffled, then the resulting sequence, X_n, belongs to $\mathcal{A}_n$.

Proof: Let n_1, n_2 denote the numbers of 0's and 1's in X_U, respectively, and let m_1, m_2 denote the numbers of 0's and 1's in X_L, respectively, where $0 \leq n_1, n_2, m_1, m_2 \leq n/2$. If $n_1 \leq m_1$, then X_n must start with n_1 multiples of 00, followed by $(m_1 - n_1)$ multiples of 10, and that followed by m_2 multiples of 11, which is an element of $\mathcal{A}_n$. If $n_1 > m_1$, then X_n must start with m_1 multiples of 00, followed by $(n_1 - m_1)$ multiples of 01, and that followed by n_2 multiples of 11, which is also an element of $\mathcal{A}_n$||.

Example 1: To see this statement, let $X_U = 1111$ and $X_L = 0001$. Then shuffling the concatenation of X_U and X_L gives 101010/11, which belongs $\mathcal{A}_8$.||

Definition 2: A binary sequence is called clean-sorted if its elements are all identical; i.e., they are either all 0 or all 1. ||

Theorem 2: Let Y_U, and Y_L denote the upper and lower halves of outputs after the first stage of $n/2$ comparators in the balanced merging block of the network in Fig. 4(b). For any binary sequence Z in $\mathcal{A}_n$ that enters the inputs of the balanced merging block, one of Y_U and Y_L must be clean-sorted, and the other must contain a binary sequence that belongs to $\mathcal{A}_{n/2}$.

Proof: Consider and arbitrary binary sequence Z in $\mathcal{A}_n$. Let Z_a, Z_b and Z_c denote the three parts of Z, respectively, as given in Definition 1, and let k_a, k_b and k_c denote the numbers of elements in Z_a, Z_b and Z_c, in that order. We have $0 \leq k_a, k_b, k_c \leq n$, and consider two cases.

Case (1): If $k_b = 0$, then, obviously, one of Y_U and Y_L must be clean-sorted.

Case (2): If $k_b \neq 0$, then we consider two cases.

Case (2.1): If $k_a \geq \frac{n}{2}$ and $k_c < \frac{n}{2}$, then Y_U must be all 0's and Y_L must contain a binary sequence that belongs to $\mathcal{A}_{n/2}$. Likewise, if $k_a < \frac{n}{2}$ and $k_c \geq \frac{n}{2}$, then Y_L must be all 1's and Y_U must contain a binary sequence that belongs to $\mathcal{A}_{n/2}$.

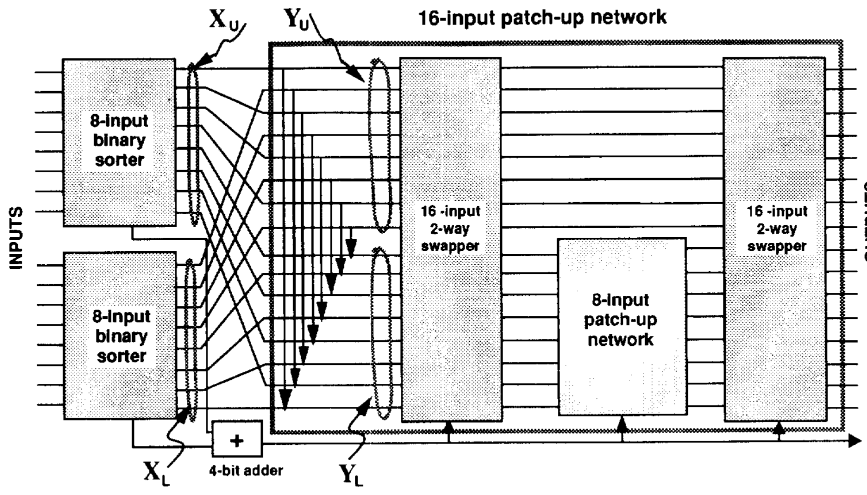


Fig. 5. A 16-input adaptive binary sorting network using prefix adder scheme.

Case (2.2): If both $k_a < \frac{n}{2}$ and $k_c < \frac{n}{2}$, then the elements of Z_b cannot all be contained in one half of the inputs of the merging network. Let Z_{bu} and Z_{bl} be the two parts of Z_b in the upper and lower halves of inputs. Let k_{bu} and k_{bl} denote the numbers of elements in Z_{bu} and Z_{bl}, respectively. We consider three cases.

Case (2.2.1): If $k_{bl} = k_{bu}$, then after the first stage of comparators, the 1's are interchanged with 0's, and Y_U must be all 0's, and Y_L must be all 1's.

Case (2.2.2): If $k_{bl} > k_{bu}$, then after the first stage of comparators, Y_U must be all 0's, and Y_L must contain a binary sequence that belongs to $\mathcal{A}_{n/2}$.

Case (2.2.3): If $k_{bl} < k_{bu}$, then after the first stage of comparators, Y_L must be all 1's, and Y_U must contain a binary sequence that belongs to $\mathcal{A}_{n/2}$.||

Example 2: To illustrate this statement, consider the sequence obtained in Example 1, i.e., 101010/11. Let this be the value of Z. After subjecting Z to the merging block, we obtain $Y_U = 1000$ and $Y_L = 1111$. Thus, one of Y_U and Y_L is clean-sorted, and the other belongs to $\mathcal{A}_4$||.

Given Theorem 2, the balanced merging block can be simplified by devising a circuit that detects which half of outputs is clean-sorted after the first stage of $n/2$ comparators in the balanced merging block, and then merges only that half that is not sorted. We can construct this circuit, which we call a *patch-up network*, recursively, as shown in Fig. 5 for $n = 16$, because by Theorem 2, the unsorted half belongs to $\mathcal{A}_{n/2}$. The detection of the unsorted half of outputs is carried out by a simple lg n-bit prefix adder that gives the count of the number of 1's in the entire input sequence. It does this by recursively adding the numbers of 1's in the two half-size input sequences as indicated in the figure.

The patch-up network is recursively constructed from a half-size patch-up network, one stage of comparators in a balanced merging block, and some swapping networks. The comparators move the 0's in the lower half up and the 1's in the upper half down whenever the compared bits are different. As implied by Theorem 2, after the comparator stage, at least one half of outputs will be clean-sorted, and the other half must be a sequence that belongs to $\mathcal{A}_{n/2}$. By examining the most significant bit of the result from the prefix adder, we can determine if the number of 1's is greater than or equal to $n/2$. If so, then the lower half of outputs must be clean-sorted, and we have the upper half left unsorted. Otherwise, the lower half of outputs must be unsorted, and the upper half must be clean-sorted.

The selection of the unsorted half of outputs is carried out by an n-input, two-way swapper. The swapper network uses the most significant bit of the sum from the prefix adder as a select input to channel the unsorted half of outputs to the next level of the patch-up network, which, by induction, is assumed to sort any binary sequence that belongs to $\mathcal{A}_{n/2}$. If its select input is 0, then the inputs are connected to the outputs straight across. If it is 1, then the upper half of inputs is connected to the lower half of outputs, and the lower half of inputs is connected to the upper half of outputs.

Once the unsorted half is selected and sorted by the patch-up network, then, if the select input of the two-way swapper in the last stage is 1, the outputs of the patch-up network are switched to the upper half of the network's outputs; otherwise, they are connected to its lower half of outputs. Given that the half-size patch-up network sorts its inputs onto its outputs, it is easy to verify that the swapper networks operating as described will produce a sorted sequence.

Corollary: The network in Fig. 5 (its n-input version) sorts any binary sequence of n elements in ascending order.

Proof: The proof immediately follows from Theorems 1 and 2.||

Let $C(n)$ and $D(n)$ denote the bit-level cost and depth of the network, respectively. Directly from Fig. 5, we have the following equation:

$$C(n) = 2C(n/2) + C_a(\lg n) + C_p(n) \qquad (1)$$

$$D(n) = D(n/2) + D_a(\lg n) + D_p(n), \qquad (2)$$

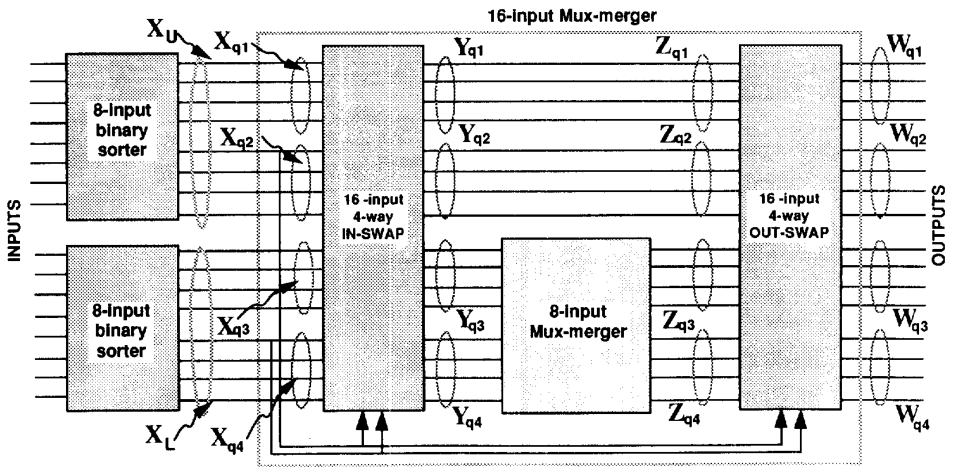


Fig. 6. A 16-input adaptive binary sorting network using multiplexed merging scheme.

where $C_a(\lg n)$ and $D_a(\lg n)$ denote the cost and depth of a $\lg n$-bit prefix adder, and $C_p(n)$ and $D_p(n)$ denote the cost and depth of the patch-up network. The cost and depth of a $\lg n$-bit prefix adder are $3 \lg n$ and $2 \lg \lg n$, respectively [5]. The cost and depth of the patch-up network can be computed from its recursive construction as follows:

$$C_p(n) = 3n/2 + C_p(n/2) \quad (3)$$

$$D_p(n) = 3 + D_p(n/2), \quad (4)$$

where the $3n/2$ term accounts for the bit-level cost of the comparators, and the two-way swappers. The solutions of these recurrences with $C_p(2) = 1, D_p(2) = 1$ give $C_p(n) \leq 3n, D_p(n) \leq 3 \lg n$.

Substituting the cost and depth expressions for the prefix adder and patch-up network into (1) and (2), and solving the recurrences with $C(2) = 1$ and $D(2) = 1$, shows that the network in Fig. 5 has $3n \lg n + O(\lg^2 n)$ cost and $3 \lg^2 n + 2 \lg n \lg \lg n$ depth, respectively.

B. Network 2 (Mux-Merger Binary Sorter)

Although the binary sorter just described has $O(n \lg n)$ cost, the use of a prefix adder could make its implementation cumbersome. Our second binary sorting network, called a *mux-merger* binary sorter, eliminates the need for a prefix adder.

Definition 3: A binary sequence is called bisorted if each of its two halves is sorted.||

Theorem 3: If a bisorted binary sequence is cut into four equal-size subsequences, then at least two of the subsequences will be clean-sorted, and the other two, when concatenated, will form a bisorted sequence.

Proof: Let X_n denote any bisorted binary sequence of size n. Let X_{q1}, X_{q2}, X_{q3} and X_{q4} denote the four quarters of X_n from top to bottom, respectively, and let X_U and X_L denote the upper and lower halves of X_n. (Refer to Fig. 6.) The proof of the statement requires checking out four cases that are identified with the binary values of the uppermost elements of X_{q2} and X_{q4}. If there are more 0's than 1's in X_U, then the uppermost element of X_{q2} must be 0, X_{q1} must contain all 0's, and X_{q2} must be a sorted sequence of size $n/4$. On the other hand, if the uppermost element of X_{q2} is 1, then there are more 1's in X_U, and hence X_{q2} must contain all 1's, and X_{q1} must be a sorted sequence of size $n/4$. Similar statements hold for X_{q3} and X_{q4}. Therefore, at least two of X_{q1}, X_{q2}, X_{q3} and X_{q4} must be clean-sorted, and the other two must form a bisorted sequence when concatenated.||

Example 3: To illustrate this theorem, consider the bisorted sequence 0001/0001. Cutting it into four equal-size subsequences 00, 01, 00, 01 reveals that two of the four subsequences are clean-sorted, and the other two, when concatenated, give 0101, which is a bisorted sequence. ||

If a binary sequence of n inputs is bisorted using two $n/2$ input binary sorters, then Theorem 3 provides a way to identify the two quarters that form a bisorted sequence of size $n/2$ when concatenated, as well as the all-0 or all-1 quarters. The network in Fig. 6 is constructed based on this fact, where the mux-merger, enclosed by the rectangle in dash lines, merges the bisorted sequence at the outputs of the two sorters into a sorted sequence.

Table I lists the possible patterns of bisorted sequences and shows how the mux-merger selects the quarter-size subsequences for each pattern.[5] The variable entries in the table denote various points in the network, as marked in Fig. 6. As described in the proof of Theorem 3, the middle two bits of the two sorted halves of outputs X_U and X_L can be one of four binary patterns, 00, 01, 10, 11. For each of these, the patterns of X_{q1}, X_{q2}, X_{q3}, and X_{q4} are uniquely determined, and the selections of the quarters can be made accordingly, as shown in the table. Furthermore, because the concatenated two quarters form a bisorted sequence, the same mux-merge process can be applied recursively. The selections described in the table can be realized by using n-input, four-way IN-SWAP and OUT-SWAP circuits, with their select control described as in the table.

[5]The symbol * used in the table denotes a concatenation operator.

TABLE I
BEHAVIOR OF MUX-MERGER

Select inputs	Input pattern	IN-SWAP select	OUT-SWAP select
00	X_{q1} and X_{q3} are all 0's, $X_{q2} * X_{q4}$ is bisorted	X_{q1} to Y_{q1}, X_{q2} to Y_{q3}, X_{q3} to Y_{q2}, and X_{q4} to Y_{q4}	Z_{q1} to W_{q1}, Z_{q2} to W_{q2}, Z_{q3} to W_{q3}, and Z_{q4} to W_{q4}
01	X_{q1} is all 0's, X_{q4} is all 1's, and $X_{q2} * X_{q3}$ is bisorted	X_{q1} to Y_{q1}, X_{q2} to Y_{q3}, X_{q3} to Y_{q4}, and X_{q4} to Y_{q2}	Z_{q1} to W_{q1}, Z_{q2} to W_{q4}, Z_{q3} to W_{q2}, and Z_{q4} to W_{q3}
10	$X_{q1} * X_{q4}$ is bisorted, X_{q2} is all 1's, and X_{q3} is all 0's	X_{q1} to Y_{q3}, X_{q2} to Y_{q2}, X_{q3} to Y_{q1}, and X_{q4} to Y_{q4}	Z_{q1} to W_{q1}, Z_{q2} to W_{q4}, Z_{q3} to W_{q2}, and Z_{q4} to W_{q3}
11	$X_{q1} * X_{q3}$ is bisorted, X_{q2} and X_{q4} are all 1's	X_{q1} to Y_{q3}, X_{q2} to Y_{q2}, X_{q3} to Y_{q4}, and X_{q4} to Y_{q1}	Z_{q1} to W_{q3}, Z_{q2} to W_{q4}, Z_{q3} to W_{q1}, and Z_{q4} to W_{q2}

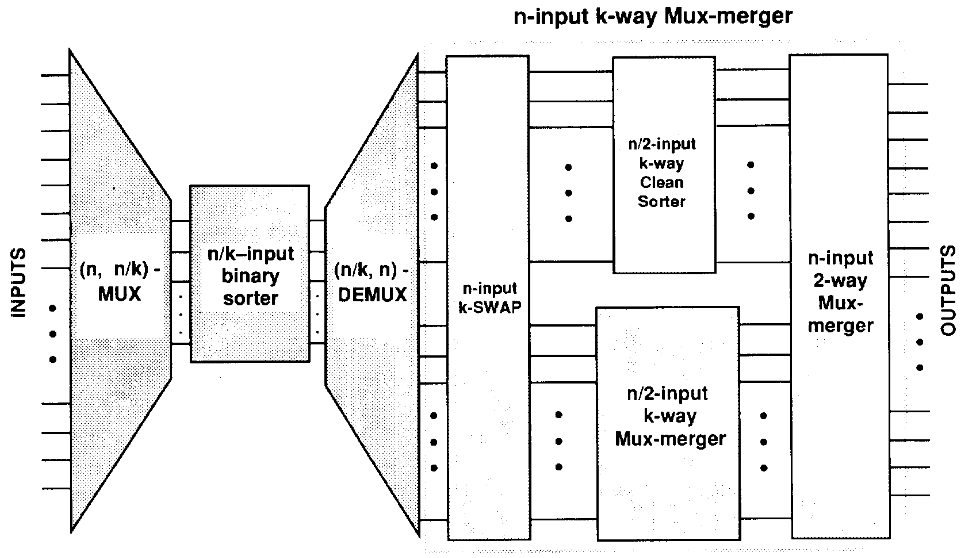


Fig. 7. An n-input adaptive binary sorting network using a time multiplexing scheme and a k-way mux-merger.

The entire sorting network can be constructed by using IN-SWAP and OUT-SWAP circuits if the half-size sorters in Fig. 6 are recursively replaced by the same sorter construction. The cost and depth of this binary sorting network construction can be expressed as follows:

$$C(n) = 2C(n/2) + C_m(n) \qquad (5)$$

$$D(n) = D(n/2) + D_m(n), \qquad (6)$$

where $C(n)$ and $D(n)$ denote the cost and depth of the entire network, respectively, and $C_m(n)$ and $D_m(n)$ denote the cost and depth of the mux-merger. Given that an n-input IN-SWAP or OUT-SWAP circuit exacts n cost and unit delay, and given that it is easy to check that if the mux-merger is constructed recursively, then $C_m(n) = 4n$ and $D_m(n) = 2 \lg n$. Solving the above recurrences with $C(2) = 1$, and $D(2) = 1$ gives $C(n) = 4n \lg n$ and $D(n) = 2 \lg n$.

C. Network 3 (Fish Binary Sorter)

The binary sorters described earlier both have $O(n \lg n)$ cost. To further reduce the cost, the network shown in Fig. 7 (called the fish binary sorter because of its resemblance to fish) can be used. In this network, we use a time-multiplexing scheme that was proposed in [15] to obtain compact permutation networks. The idea is to multiplex the binary inputs, in time, through a binary sorting network with a smaller number of inputs. Any binary sorting network including those described in the previous subsection can be used in this kind of multiplexed sorting. The input sequence is first arbitrarily divided into k groups of n/k elements. Each group of inputs is run through an $(n,\ n/k)$-multiplexer ($(n,\ n/k)$-MUX) sequentially, and sorted by an n/k-input binary sorter. The sorted n/k-element sequences are then moved through an $(n/k,\ n)$-demultiplexer ($(n/k,\ n)$-DEMUX) circuit to the inputs of an n-input k-way merging network, where the first n/k inputs of the merging network are occupied by the first sorted n/k-element sequence, the second n/k inputs are occupied by the second sorted n/k-element sequence, and so on.

All that remains to be done is to give a construction for an n-input, k-way merger. For this, we first generalize the notion of a bisorted binary sequence.

Definition 4: A binary sequence is called clean k-sorted if it consists of k equal-size sorted binary subsequences. For

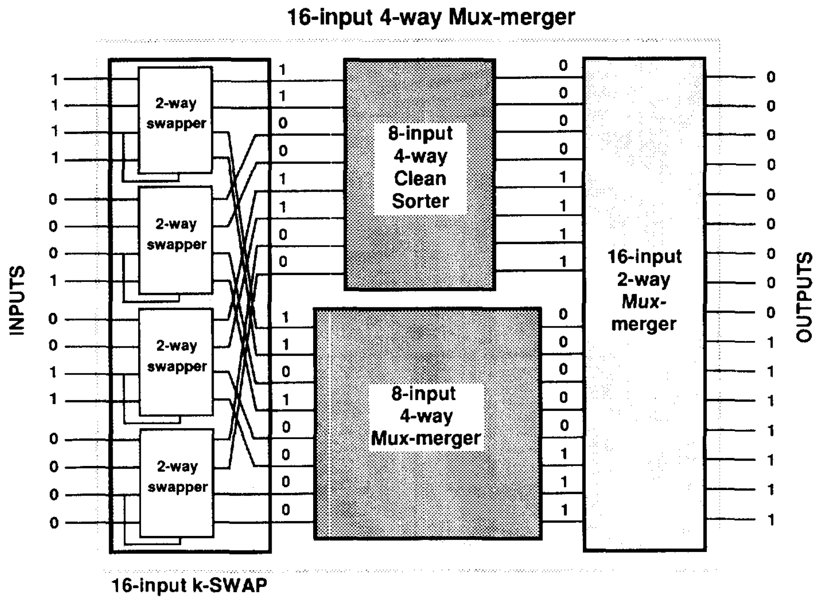

16-input 4-way Mux-merger
INPUTS
2-way swapper
8-input 4-way Clean Sorter
8-input 4-way Mux-merger
16-input 2-way Mux-merger
OUTPUTS
16-input k-SWAP

Fig. 8. An example showing the operations of a 16-input four-way mux-merger.

example, for $k = 4$, 1111/0001/0011/0111 is a 4-sorted sequence.||

Definition 5: A k-sorted binary sequence is called k-sorted if each of its k equal-size sorted binary subsequences is clean-sorted; i.e., each contains only 0's or only 1's. For example, for $k = 4$, 1111/0000/0000/1111 is a clean 4-sorted sequence. ||

Theorem 4: If, in a k-sorted binary sequence, each of the k equal-size sorted binary subsequences is cut into two half-size subsequences, then at least k of the half-size subsequences will be clean-sorted, which, if concatenated, will form a clean k-sorted sequence, and the remaining k half-size subsequences, when concatenated, will form a k-sorted sequence.

Proof: The proof is a direct extension of the proof of Theorem 3.||

Example 4: To illustrate the theorem, consider the 4-sorted sequence, 1111/0001/0011/0111. Cutting each subsequence in half gives 11, 11, 00, 01, 00, 11, 01, 11. Of the eight subsequences, six (more than half) are clean-sorted. Putting 11, 00, 11, 11 together, we get a clean 4-sorted sequence, and the other four form a sequence 11/01/00/01 that is 4-sorted.

Using Theorem 4, we can then extend the two-way mux-merger to a k-way mux-merger. In an n-input, k-way mux-merger, a k-sorted input sequence of size n, which consists of k-sorted subsequences each of size n/k, is first separated into two $n/2$-bit sequences by performing what is called a k-SWAP operation using k n/k input two-way swappers. By connecting the middle bit of each of the k sorted subsequences as the select signal, each of these k n/k-input two-way swappers sends the clean-sorted halves up and the other halves down. As a result, the upper $n/2$ outputs of the n input k-swapper consist of k clean-sorted subsequences, each of size $n/2k$, and thus form a clean k-sorted sequence. Also, the lower $n/2$ outputs consist of k sorted subsequences of size $n/2k$, and form a k-sorted sequence of size $n/2$.

Since the upper $n/2$-size sequence is clean k-sorted consisting of k clean-sorted subsequences of size $n/2k$, by sorting each leading bit of these subsequences and then sending each subsequence to its corresponding sorted position, a sorted output can be obtained. These steps can be carried out by employing a k-input sorter, an $(n/2, n/2k)$-multiplexer, and an $(n/2k, n/2)$-demultiplexer. These circuits are collectively referred to as an $n/2$-input, k-way clean sorter in Fig. 7.

Assuming that the lower $n/2$ outputs of the k-swapper can be merged recursively, we can use an n-input two-way mux-merger at the last stage to merge these two sorted $n/2$-bit sequences and obtain a sorted output. As an example, Fig. 8 illustrates the operation of an n-input k-way mux-merger for $n = 16$ and $k = 4$. The operation of the $n/2$-input, k-way clean sorter is also illustrated in Fig. 9, for $n = 16$ and $k = 4$.

The cost $C(n, k)$ and depth $D(n, k)$ of this binary sorting network can be expressed as follows:

$$C(n,k) = C_d(n, n/k) + C_s(n/k) + C_{km}(n,k), \quad (7)$$
$$D(n,k) = D_d(n, n/k) + D_s(n/k) + D_{km}(n,k), \quad (8)$$

where $C_d(n, n/k)$ and $D_d(n, n/k)$ denote the cost and depth of the $(n, n/k)$-multiplexer and $(n/k, n)$-demultiplexer circuits combined together; $C_s(n/k)$ and $D_s(n/k)$ denote the cost and depth of the n/k-input binary sorter; and $C_{km}(n,k)$ and $D_{km}(n,k)$ denote the cost and depth of the n-input k-way mux-merger. It follows from Fig. 7 that the following is true:

$$\begin{aligned} C_{km}(n,k) &= C_{\text{SWAP}}(n) + C_{cs}(n/2,k) \\ &\quad + C_{km}(n/2,k) + C_m(n), \quad (9) \\ D_{km}(n,k) &= D_{\text{SWAP}}(n) + \max(D_{cs}(n/2,k), \\ &\quad D_{km}(n/2,k)) + D_m(n), \end{aligned}$$
$$(10)$$

where $C_{\text{SWAP}}(n)$ and $D_{\text{SWAP}}(n)$ denote the cost and depth of the n-input k-swapper; $C_{cs}(n/2,k)$ and $D_{cs}(n/2,k)$ denote the cost and depth of the $n/2$-input k-way clean-sorter; and $C_m(n)$ and $D_m(n)$ denote the cost and depth of the n-input two-way mux-merger, respectively.

Since an n-input k-swapper can be constructed with k n/k-input two-way swappers, its cost $C_{\text{SWAP}}(n)$ is $n/2$, and its depth $D_{\text{SWAP}}(n)$ is 1. As illustrated in Fig. 9, the $n/2$-input k-way clean-sorter can be constructed with a k-input binary sorter, sorting the leading bits of each of the k clean-sorted subsequences, and with a dispatching circuit to send each subsequence to its sorted position. This circuit uses an $(n/2, n/2k)$-multiplexer, and $(n/2k, n/2)$-demultiplexer and a $(k,1)$-multiplexer, and thus exacts $n+k$ cost and $3 \lg k$ depth. If the k-input binary sorter is realized by the mux-merger binary sorter, then its cost and depth will be $4k \lg k$ and $2 \lg^2 k$, respectively. Finally, the cost and depth of the n-input, two-way mux-merger are $4n$ and $2 \lg n$, respectively. Substituting all these into (9) and (10) yields the following equations:

$$C_{km}(n,k) = n/2 + 4k \lg k + n + k + C_{km}(n/2,k) + 4n \quad (11)$$
$$= \frac{11n}{2} + 4k \lg k + k + C_{km}(n/2,k) \quad (12)$$
$$D_{km}(n,k) = 1 + \max((2 \lg^2 k + 3 \lg k), D_{km}(n/2,k)) + 2 \lg n \quad (13)$$
$$\le 1 + D_{km}(n/2,k) + 2 \lg n. \quad (14)$$

Now suppose that the n-input, k-way mux-merger is constructed by recursively decomposing its $n/2$-input, k-way mux-merger component until it reduces to have k inputs, and the k-input, k-way merger is realized by a k-input mux-merger binary sorter. This amounts to solving the above recurrences with the boundary condition $C_{km}(k,k) = 4k \lg k$ and $D_{km}(k,k) = 2 \lg^2 k$, which yields the following equation:

$$C_{km}(n,k) = 11n - 11k + k \lg \frac{n}{k} + 4k \lg k \lg \frac{n}{k} + 4k \lg k \quad (15)$$

$$D_{km}(n,k) \le \lg \frac{n}{k} + 2 \lg n \lg \frac{n}{k} + 2 \lg^2 k. \quad (16)$$

Substituting these into (7) and (8), we find the following:

$$C(n,k) \le 2n + 4\frac{n}{k} \lg \frac{n}{k} + 11n + k \lg \frac{n}{k} + 4k \lg k \lg \frac{n}{k} + 4k \lg k \quad (17)$$

$$D(n,k) \le 2 \lg k + 2 \lg^2 \frac{n}{k} + \lg \frac{n}{k} + 2 \lg n \lg \frac{n}{k} + 2 \lg^2 k, \quad (18)$$

and minimizing these expressions with respect to k, we obtain the following equations:

$$C(n, \lg n) \le 17n + 5 \lg^2 n \lg \lg n + 4 \lg n \lg \lg n = O(n), \quad (19)$$

$$D(n, \lg n) \le 2 \lg \lg n + 2 \lg^2 n + \lg n + 2 \lg^2 n \quad (20)$$
$$= O(\lg^2 n). \quad (21)$$

It is obvious that the sorting time, $T(n,k)$, can be similarly determined as follows:

$$T(n,k) = k(D_d(n,n/k) + D_s(n/k)) + D_{km}(n,k) \quad (22)$$
$$= 2k \lg k + 2k \lg^2 \frac{n}{k} + \lg \frac{n}{k} + 2 \lg n \lg \frac{n}{k}, \quad (23)$$

and letting $k = \lg n$ gives the following:

$$T(n, \lg n) = O(\lg^3 n). \quad (24)$$

The sorting time can be reduced to $O(\lg^2 n)$ by noting that the k groups of n/k inputs can be pipelined through the n/k-input sorting network. In this case, the sorting network is viewed as a $\lg^2 n/k$ segment pipeline, where each segment is a constant fanin, unit delay circuit. Therefore, the sorting time $T_{\text{pip}}(n,k)$ is given by the following equation:

$$T_{\text{pip}}(n,k) = O(\lg^2 \frac{n}{k}) + O(k) + O(\lg k) + O(\lg n \lg n/k), \quad (25)$$

where the $O(\lg^2 (n/k)$ and $O(\lg k)$ terms account for the time for the first group of n/k elements to exit the pipeline, and the $O(k)$ term accounts for the time that the remaining $k-1$ groups of n/k elements need to get through the pipeline. The $O(\lg n \lg k)$ term is the merging time. Letting $k = \lg n$ gives $T_{\text{pip}}(n, \lg n) = O(\lg^2 n)$.

We note that a time-multiplexed network version of the columnsort algorithm proposed by Leighton in [14] has the same bit-level cost of $O(n)$. The columnsort algorithm, where n inputs are divided into s columns and r rows, consists of eight steps, four of which sort columns of inputs; the other four steps rearrange the sorted columns of inputs. It was mentioned in [14] that a columnsort sorting network with $O(n)$ processors could be obtained if one uses an $n/\lg n$-input AKS sorter to sort each of $\lg n$ columns of $n/\lg n$ elements. Because the AKS network is not practical, one can use an $n/\lg^2 n$-input Batcher's sorter to sort each of $\lg^2 n$ columns of $n/\lg^2 n$ elements and also obtain a sorting network with $O(n)$ cost. In this version of columnsort network, however, the inputs must be mutliplexed into smaller size sorters, and the outputs from the sorters must be demultiplexed. This requires additional circuits whose cost would be comparable to the cost of the (n,k)-multiplexer and (k,n)-demultiplexer used in our fish binary sorter. The sorting time of this time-multiplexed columnsort network would be $O(\lg^4 n)$ without pipelining and $O(\lg^2 n)$ with inputs pipelined. We note that the pipelined version of this network requires that the data be separately pipelined through each of the four sorters in its construction. In contrast, our last network construction needs to pipeline the inputs through only a single $n/\lg n$-input sorter.

It should also be noted that without time-multiplexing, a practical binary columnsort network, e.g., one using Batcher's sorters, would require $\lg^2 n$ $(n/\lg^2 n)$-input Batcher's sorters in its construction, resulting in a bit-level cost of $O(n \lg^2 n)$. In contrast, the mux-merger binary sorter described in the previous subsection would have only a bit-level cost of $O(n \lg n)$.

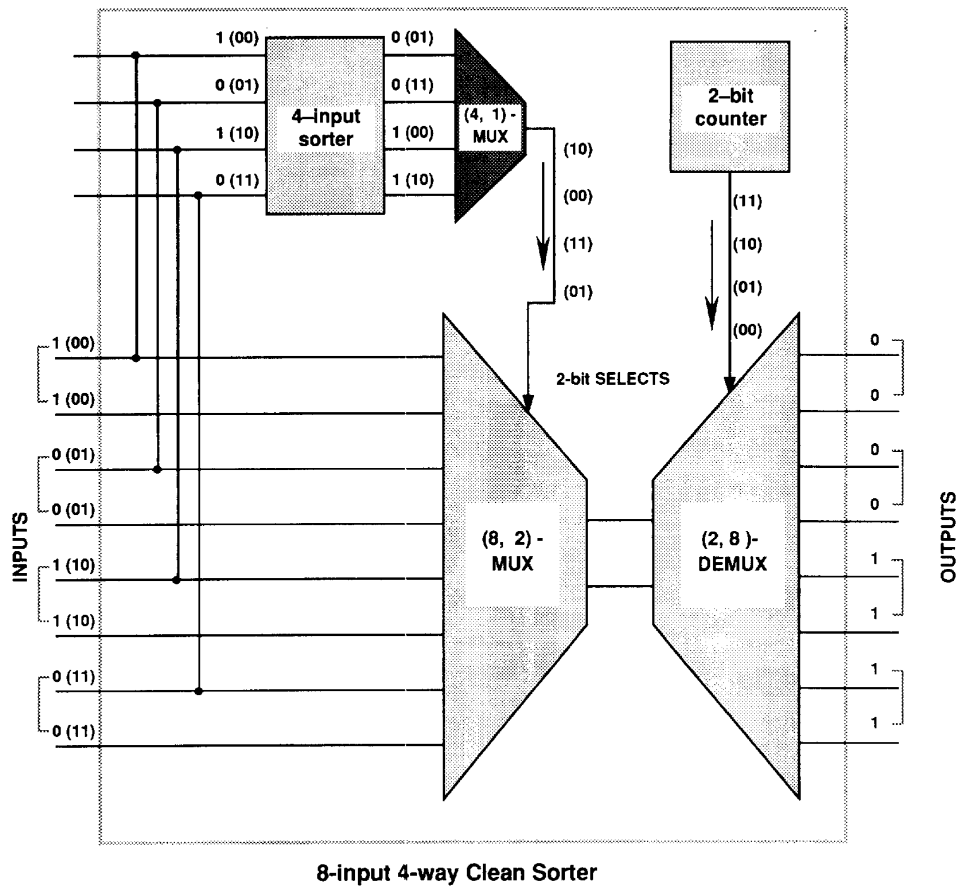


Fig. 9. An example showing the operations of an eight-input four-way clean sorter.

IV. Concentrators and Permutation Networks Using Binary Sorters

Concentration and permuting are two communication problems that frequently arise in parallel computations Both problems can be solved efficiently by using binary sorters. Formally, an (n, m)-concentrator is a network with n inputs and m outputs, $m \leq n$, that can map any $r \leq m$ of its inputs to some r distinct outputs, e.g., the first r outputs, $1 \leq r \leq m$. It should be easy to see that a binary sorter does form an (n, n)-concentrator. All that is needed is to tag the inputs to be concentrated with 0's and tag the remaining inputs with 1's.

Concentrators have been reported in the literature. The expander-based constructions reported in [2], [10], [16], [21], [22], provide $O(n)$ cost concentrators, but their concentration time is not known. The ranking tree-based constructions given in [11], [13], exact $O(n \lg^2 n)$ cost. The prefix-adder and mux-merger binary sorters described in Section III give (n, n)-concentrators with $O(n \lg n)$ cost and $O(\lg^2 n)$ depth. The fish binary sorter provides a time-multiplexed concentrator with $O(n)$ cost and $O(\lg^2 n)$ concentration time. The only other time-multiplexed concentrator that matches these cost and time complexities is the network version of the columnsort algorithm [14], as we mentioned in the previous section.

Binary sorters can also be used to form permutation networks. Much work on permutation networks has been reported in the literature. The well-known Beneš network [4] has $O(n \lg n)$ cost and $O(\lg n)$ depth, though realizing arbitrary permutations on this network requires $O((\lg^4 n)/\lg \lg n)$ time on an $n \lg n$ processor, perfect shuffle, or cube-connected parallel computer [18]. Batcher's sorting networks [3] and those that appeared in [7], [13], can also be used for permutation switching, but they require $O(n \lg^3 n)$ cost and $O(\lg^3 n)$ permutation time in bit-level.

More recently, Jan and Oruç proposed a permutation network with $O(n \lg^2 n)$ cost and $O(\lg^2 n \lg \lg n)$ permutation time in bit-level [11]. This permutation network, called a radix permuter, is recursively constructed from a distributor, two concentrators, and two half-size radix permuters. Our binary sorters can be used to replace the distributor and two concentrators in this construction, because by sorting the leading bits in the destination address, a binary sorter can distribute the inputs to the upper and lower half-size radix permuters, as shown in Fig. 10. If the fish binary sorter is used in this construction, then the cost $C_{rp}(n)$ and depth $D_{rp}(n)$ of this permutation network are given as follows:

$$C_{rp}(n) = O(n) + 2C_{rp}(n/2) = O(n \lg n) \qquad (26)$$

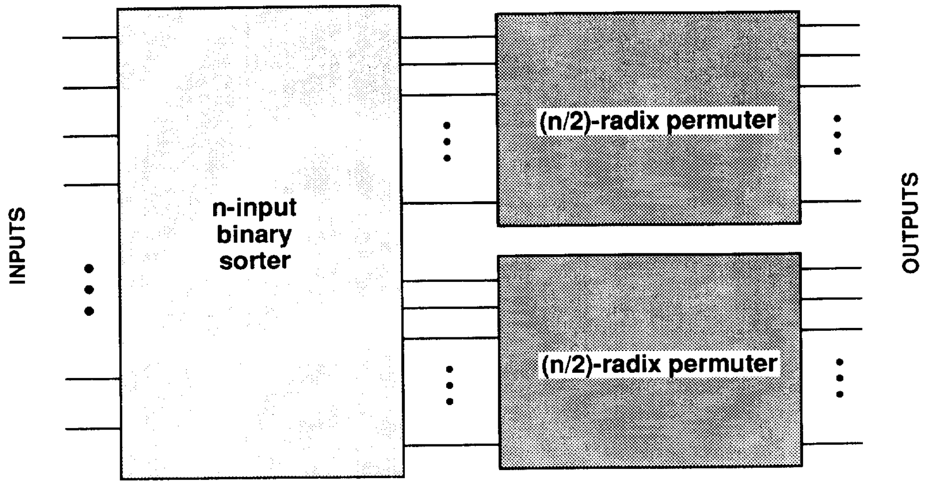


Fig. 10. An n-input permutation network using adaptive binary sorters.

TABLE II
COMPLEXITIES OF VARIOUS PERMUTATION NETWORK DESIGNS IN BIT LEVEL

Construction	Cost	Depth	Permutation time
[4], [18]	$O(n \lg^2 n)$	$O(\lg n)$	$O((\lg^4 n)/\lg \lg n)$
[3]	$O(n \lg^3 n)$	$O(\lg^3 n)$	$O(\lg^3 n)$
[13]	$O(n \lg^3 n)$	$O(\lg^3 n)$	$O(\lg^3 n)$
[11]	$O(n \lg^2 n)$	$O(\lg^2 n \lg \lg n)$	$O(\lg^2 n \lg \lg n)$
This paper	$O(n \lg n)$	$O(\lg^3 n)$	$O(\lg^3 n)$

$$D_{rp}(n) = O(\lg^2 n) + D_{rp}(n/2) = O(\lg^3 n). \quad (27)$$

The permutation time of this network is the same as its depth.

It must be pointed out that because the fish binary sorter relies on time-multiplexing, the radix permuter that is obtained by replacing concentrators with fish binary sorters is a packet-switched network. On the other hand, radix permuters obtained by replacing concentrators with either prefix binary sorter or mux-merger binary sorter give a circuit-switched permutation network.

Table II lists the complexities of this permutation network along with those of other permutation networks reported earlier. The cost of the Beneš network includes the cost of the $O(n \lg n)$ processors in its routing model [18], where the lg n factor in the cost expression accounts for the bit-level cost of each processor. The table reveals that the network given in this paper has the smallest order of cost complexity, and that its depth and permutation time match the depths and permutation times of those networks given in [3] and [13], and also that they are slightly higher than the depth and permutation time of the network obtained in [11].

V. CONCLUDING REMARKS

The paper has introduced adaptive sorting networks and presented an in-depth analysis of sorting binary sequences on such networks. The results are rewarding in that under the adaptive sorting network model, we were able to obtain $O(n \lg n)$ bit-level cost binary sorting networks, and $O(n)$ bit-level cost time-multiplexed binary sorting networks, all with $O(\lg^2 n)$ bit-level sorting time.

The paper also described the first n-input permutation network with $O(n \lg n)$ bit-level cost and $O(\lg^3 n)$ bit-level routing time. A permutation network with $O(n \lg^2 n)$ bit-level cost, but with a much simpler design, can also be obtained by using the mux-merger sorter described in the paper.

An attractive feature of all of the network constructions in the paper is that the constants in the cost, depth, and time complexity expressions are very small (≤ 17) as compared to the constants in the complexities of expander-based constructions, such as the AKS network. A worthwhile extension of these results will be to construct a binary sorter with $O(n)$ cost and $O(\lg n)$ sorting time where the constants in the order of complexity expressions are small.

ACKNOWLEDGMENT

The authors thank the anonymous referees for their constructive comments.

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, "Sorting in $c \log n$ parallel steps," *Combinatorica*, vol. 3, pp. 1–19, Jan. 1983.
[2] N. Alon, "Eigenvalues and expanders," *Combinatorica*, vol. 6, pp. 83–96, 1986.
[3] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Conference*, 1968, pp. 307–314.
[4] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965.
[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.
[6] T. H. Cormen, "Concentrator switches for routing messages in parallel computers," M.S. thesis, Massachusetts Inst. of Technol., 1986.
[7] B. Douglass and A. Yavuz Oruç, "Self-routing and route balancing in connection networks," Tech. Rep. UMIACS-TR-90-32,CS-TR-2421, Inst. for Advanced Comput. Studies, Univ. of Maryland, College Park, MD, 1990.
[8] M. Dowd, Y. Perl, L. Rudolph, and M. Saks, "The balanced sorting network," in *Proc. 2nd Ann. ACM Symp. Principles of Distrib. Computing*, 1983, pp. 161–172.

[9] M. Dowd, Y. Perl, L. Rudolph, and M. Saks, "The periodic balanced sorting network," *J. ACM*, vol. 36, pp. 738–757, Oct. 1989.

[10] O. Gabber and Z. Galil, "Explicit constructions of linear sized superconcentrators," *J. Comput. Syst. Sci.*, vol. 22, pp. 407–420, 1981.

[11] C. Y. Jan and A. Yavuz Oruç, "Fast self-routing permutation switching on an asympotically minimum cost network," Tech. Rep. UMIACS-TR-91-127, CS-TR-2753, Inst. for Advanced Comput. Studies, Univ. of Maryland, College Park, MD, 1991.

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

[13] D. Koppelman and A. Yavuz Oruç, "A self-routing permutation network," *J. Parallel Distrib. Computing*, vol. 6, pp. 140–151, Aug. 1990.

[14] T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, vol. C-34, no. 4, pp. 344–354, Apr. 1985.

[15] M. Lu and A. Yavuz Oruç, "Efficient networks for realizing point-to-point assignments in parallel processors," Tech. Rep. UMIACS-TR 92-63,CS-TR-2910, Inst. for Advanced Comput. Studies, Univ. of Maryland, College Park, MD, 1992.

[16] G. A. Margulis, "Explicit constructions of concentrators," *Problems Inform. Transmission*, vol. 9, no. 4, pp. 325–332, 1973.

[17] Muller and Preparata, "Bounds to complexities of networks for sorting and switching," *J. ACM*, vol. 22, pp. 195–201, Jan. 1975.

[18] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Beneš network," *IEEE Trans. Comput.*, vol. C-31, pp. 148–154, Feb. 1982.

[19] I. Parberry, "Current progress on efficient sorting networks," Tech. Rep. CS-89-30, Dept. of Comput. Sci., Pennsylvania State Univ., University Park, PA, 1989.

[20] M. S. Paterson, "Improved sorting networks with $O(\lg n)$ depth," *Algorithmica*, vol. 5, pp. 75–92, 1990.

[21] M. S. Pinsker, "On the complexity of a concentrator," in *Proc. 7th Int. Teletraffic Congress*, 1973, pp. 318/1–318/4.

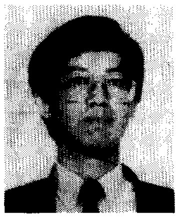
[22] N. Pippenger, "Superconcentrators," *SIAM J. Computing*, vol. 6, pp. 298–304, 1977.

[23] D. Richards, "Parallel sorting: A bibliography," *ACM SIGACT News*, vol. 18, pp. 28–48, Summer 1986.

[24] L. Rudolph, "A robust sorting network," *IEEE Trans. Comput.*, vol. C-34, no. 4, pp. 326–335, Apr. 1985.

[25] D. C. Van Voorhis, "An economical construction for sorting networks," in *Proc. AFIPS Nat. Comput. Conf.*, vol. 43, pp. 921–926, 1974.

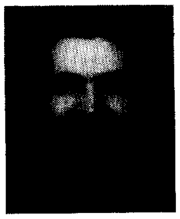
[26] I. Wegener, The Complexity of Boolean Functions. Chichester, UK: Wiley, 1987.



Minze V. Chien (S'88–M'92) received the B.S. degree in physics fro Fu Jen Catholic University, Taiwan, Republic of China, in 1980, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, MD, in 1988 and 1992, respectively.

His current research interests include interconnection networks, parallel processing, distributed systems, image processing, very large scale integration complexity, and neural networks.

Dr. Chien is a member of the IEEE Computer Society and ACM.



A. Yavuz Oruç (S'82–M'83–SM'92) received the B.Sc. degree in electrical engineering from the Middle East Technical University, Ankara, Turkey, in 1976, the M.Sc. degree in electronics from the University of Wales, Cardiff, UK, in 1978, and the Ph.D. degree from Syracuse University, Syracuse, NY, in 1983.

Since January 1988, he has been an Associate Professor in the Department of Electrical Engineering, University of Maryland, College Park, MD. Prior to joining the University of Maryland, he was on the faculty of the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY. His research interests include parallel computer and communication systems.

Dr. Oruç is a member of the IEEE Computer Society and the IEEE Information Theory Society.