June, 1983

THE BALANCED SORTING NETWORK

by

M. Dowd[1]*, Y. Perl[2,3]*, M. Saks[4]*
and L. Rudolph[3]**

DCS-TR-127

*Department of Computer Science,
Rutgers University

**Department of Computer Science
Carnegie-Mellon University

Department of Computer Center
Rutgers University
New Brunswick, New Jersey

[1]Supported in part by NSF Grant MCS-8019004

[2]On leave from Bar-Ilan University, Israel

[3]Part of this work done at University of Toronto

[4]Supported in part by NSF Grant MCS-8102448

Abstract

This paper introduces a new sorting network, called the balanced sorting network, that sorts n items in $O([\lg n]^2)$ time using $(n/2)(\lg n)^2$ comparators. Although these bounds are comparable to many existing sorting networks, the balanced sorting network possess some distinct advantages. In particular, its structure is highly regular consisting of a sequence of *identical* balanced merging networks. We prove that $\lg n$ identical merging networks are both necessary and sufficient to sort n items. We also present an explicit implementation of our network on the shuffle exchange interconnection model in which the direction of the comparitors are all identical and fixed.

1 INTRODUCTION

Sorting is one of the most extensively studied problems in computer science. This paper continues this study by introducing a new sorting network, called the balanced sorting network. This network requires $O([\lg n]^2)$ time[5] to sort input vectors of size $n=2^k$ by using $(n/2)(\lg n)^2$ comparators. Although these bounds are comparable to many existing sorting networks, the balanced sorting network possess some distinct advantages. In particular, its structure is highly regular consisting of a sequence of k *identical* balanced merging networks and thus has nice hardware implementations. Moreover, the surprising fact that a merge-sort can be accomplished with identical merging blocks (not recursive structures) is of theoretical interest.

Sorting networks of this type have been well studied. Batcher [Ba] presented two: the bitonic network and the odd-even network. (See also Knuth [K]). Hong and Sedgewick [HS] studied the relationship between both of the Batcher networks and described almost optimal networks for merging sequences of sizes n and m when n is not equal to m. Reif and Valiant [RV] presented a sorting algorithm for a CCC parallel machine (which could be viewed as a sorting network) that is based on quicksort [H] and sorts in $O(\lg n)$ time with very high probability. A surprising recent result, by Ajtai et. al [AKS], is an $O(\lg n)$ sorting network using only $O(n \lg n)$ comparators. Their result, although asymptotically superior to the one presented here, is only of theoretical interest since the constants are unacceptably large.

The network presented here is related to quicksort. In fact, the basic idea came while watching an educational film on sorting techniques [Be]. Recall that in quicksort a 'splitter' element is chosen and used to partition the elements into two sets, those larger and those smaller than the splitter. The partitioning is usually accomplished by proceeding from both ends of the vector and whenever an element in the left side is larger than the splitter it is exchanged with an element in the right side that is smaller than the splitter. A visual simulation of quicksort shows the exchanges of elements but masks the comparisons between the splitter element and the remaining elements. This leads one to wonder if the comparisons with the splitter can be eliminated and replaced with comparisons between the elements in the right side with their symmetric counterparts in the left side.

The structure of the balanced merging network is a direct consequence of this observation. Specifically, a basic unit of the network is a two input, two output comparator transforming the arbitrary order of the two input elements into nondecreasing order. Each 'phase' of the balanced

---

[5] We use the symbol "lg" to represent "$\log_2$"

merging network is composed of $n/2$ of these comparators with the first phase comparing elements $x(0)$ with $x(n-1)$,$x(1)$ with $x(n-2)$, $\cdots$,$x(n/2-1)$ with $x(n/2)$, where x is the input vector. We take the approach of an 'oblivious' algorithm in that even though the first phase does not guarantee a partition of the input into two halves, we pretend that it does and thus we continue to apply the same procedure to both halves of the output of the first phase recursively. Figure 1 depicts a balanced merging network for n = 16 elements using Knuth's [K] comparator-network representation where horizontal lines represent the input lines $x(i), 0 \le i < n$, and vertical lines represent comparisons between the elements on the corresponding input lines. Since the output of a merging network (from now on we call such a merging network a block) may not be sorted, we continue to apply these blocks until sorting is obtained. (Figure 5 shows the entire balanced sorting network for size 16.)

Each block, as its name implies, is a merging network, however, it is not easily observed exactly what is being merged. The first phase of the merging network applied to a 'recursively balanced' vector partitions the elements so that the $n/2$ smallest elements are in the first half of the vector and the $n/2$ largest elements are in the second half of the vector. Moreover, each half is recursively balanced so that each subsequent phase acts recursively to sort the input.

Vectors in which the two sequences of the elements in the odd positions and in the even positions are both sorted are shown to be recursively balanced. Thus the balanced merging network actually merges two sorted sequences (hence its name).

As a serial software sorting method, the balanced sort's worst case complexity is $O(n \lg^2 n)$, while the worst case and average case complexities of quicksort are $O(n^2)$ and $O(n \lg n)$, respectively. We still have to analyze the average complexity of our method.

The outline of the paper is as follows. After first comparing our network with those of Batcher, we then discuss some of its nonobvious properties. The section also shows that $\lg n$ blocks are necessary to sort.

The main result, that $\lg n$ blocks suffice to sort a vector of size n, is presented and proved in the section 3. In order to show that each block doubles the size of a sorted chain, we introduce the intermediate notion of a sorted 'cochain' and show that the initial 'phases' of a block merges sorted chains into sorted cochains. Another phase of the block then merges sorted cochains into sorted chains.

The third section investigates hardware implementations of the network. Stone[St] implemented

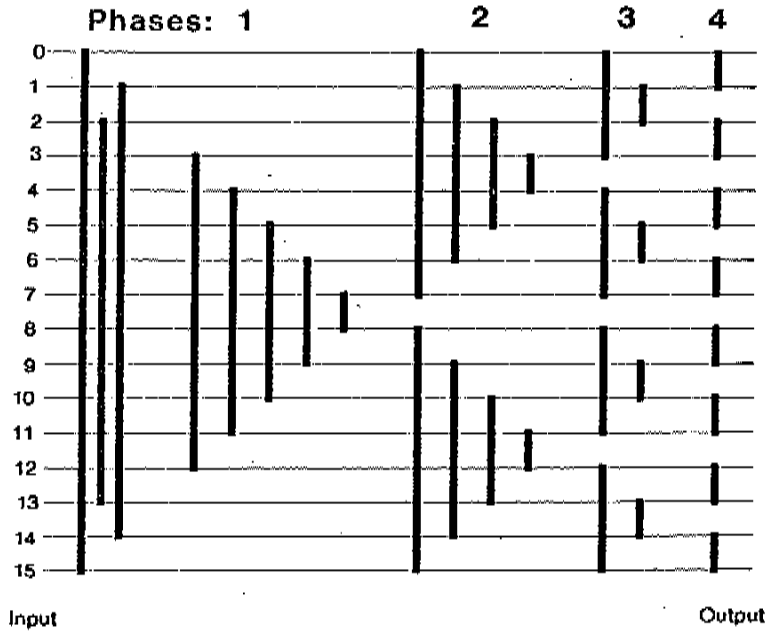


Figure 1

A Balanced Merging Block For 16 Items.

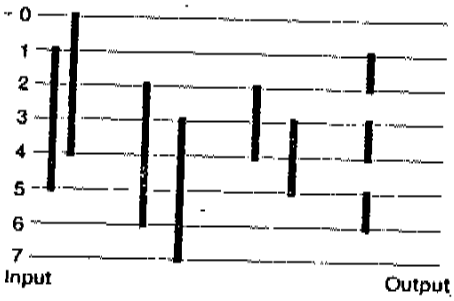


Figure 2

Odd-Even Merging Network for 8 Items.

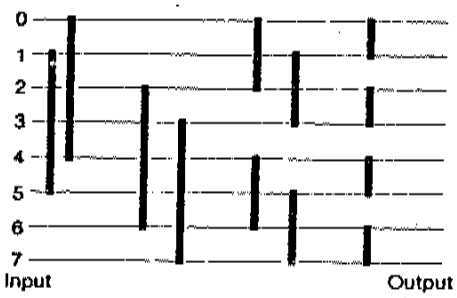


Figure 3

Bitonic Merging Network for 8 Items.

Batcher's bitonic network on the shuffle exchange interconnection model such that only one layer of comparators is required. However, in his implementation the directions of the comparators are switched during the phases (i.e., exchanging the direction of the output). We also present an explicit implementation of our network on the shuffle exchange interconnection model, however, in this implementation the direction of all the comparators is identical and fixed, and the corresponding program for a parallel computer is simpler than that for the bitonic network.

2 DISCUSSION OF THE NETWORK

In this section we discuss some of the nonobvious aspects of the balanced sort network. We compare it to those of Batcher's and show that they are not equivalent. Although our network was designed as a partition sort, it is actually a merging network. We conclude this section with an input vector of size n that requires $\lg n$ blocks to accomplish sorting. Thus, $\lg n$ blocks are necessary to sort; the following section proves that they are also sufficient.

2.1 Comparisons with Odd-Even and Bitonic Networks

At this point it may be instructive to compare our merging network to those of Batcher. In the odd-even network two increasing sequences of $n/2$ elements are merged by recursively merging the odd elements and the even elements separately and then applying a phase of $n/2-1$ comparators (in parallel) to pairs $x(1)$ and $x(2)$,$x(3)$ and $x(4),\ldots,x(n-3)$ and $x(n-2)$ yielding a sorted output. An odd-even network for merging 8 elements appears in Figure 2.

A bitonic sequence is obtained by concatenating two monotonic sequences, one increasing and one decreasing. In the bitonic merging network, the input is a bitonic sequence of $n/2$ increasing elements and $n/2$ decreasing elements. These two sequences are merged by first applying $n/2$ comparators to $x(0)$ and $x(n/2)$, $x(1)$ and $x(n/2+1),\ldots,x(n/2)$ and $x(n-1)$. This first phase partitions the elements into two bitonic sequences of the $n/2$ smaller elements and of the $n/2$ larger elements. These two bitonic sequences are further sorted by applying two bitonic merging networks of size $n/2$ to each sequence. A bitonic merging network for merging 8 elements appears in Figure 3.

Note the similarity between the bitonic merging network and our balanced merging network. In both networks the first phase compares the smallest element of the first sequence with the largest element of the second sequence, the second smallest of the first sequence with the second largest of the second sequence, etc. The effect of this first phase for both networks is partitioning the elements into two halves of the smaller elements and of the larger elements such that each half has some special structure.

Despite their similarity, there is no permutation between the input lines of the two networks, and thus these are basically two different networks. Indeed, that the differences between our network and those of Batcher are not superficial is evident from the following lemma which is satisfied by neither the odd-even nor bitonic merge networks.

Lemma 1:

i) If no exchange occurs during a block, then the input of the block is sorted.

ii) A network which sorts any input can be constructed by serially composing a finite number of blocks.

Proof:

i) A single block performs all comparisons $x(i-1)$ with $x(i)$ for $1 \leq i < n$ (among others) and thus if no exchange occurs the input must be sorted.

ii) Each exchange decreases the number of inversions (that is, pair of elements which are out of order). Since a permutation has at most $\binom{n}{2}$ inversions, (i) implies that $\binom{n}{2}$ blocks suffice to sort. □

The natural question to ask is: how many blocks are necessary and sufficient to sort any input. An example of a binary input requiring $k = \lg n$ blocks has $x(0) = x(n-1) = 1$ and $x(i) = 0,\ 1 \leq i \leq n-2$. To see this one should observe that applying one block and the first phase of the second block to this input yields $y(n/2) = y(n-1) = 1$ and $y(i) = 0, i \neq n/2, n-1$. In other words the lower half of y is full with zeros and the upper half has the same structure of the original input. An induction shows that $\lg n$ blocks are required to sort this input.

2.2 Remarks

In the next section we prove that $\lg n$ blocks are sufficient to sort any input. The main step is the proof of the following fact about the balanced merging network (which generalizes the fact that it merges two sorted sequences in the even and odd positions): if the 2^j subvectors of the input x indexed by the different congruence classes modulo 2^j are each sorted then the 2^{j-1} subvectors of the output indexed by the congruence classes mod 2^{j-1} are each sorted. Thus a block applied to an arbitrary input yields an output y such that each of the $n/2$ pairs of elements $(y(i), y(i+n/2)), 0 \leq i < n/2$ is sorted. After j blocks, each of the 2^{k-j} subvectors indexed by the congruence classes mod 2^{k-j} are sorted. In particular, after $k(= \lg n)$ blocks the output is sorted. Note that the sorted subvectors mentioned above also appear in Shell's $O(n^{3/2})$ sorting method [Sh].

One interesting point is that although the balanced merging network was designed according to the partitioning approach, its behavior resembles that of a merging network. It should be emphasized, however, that there is a fundamental difference between the sorting network consisting of $\lg n$ identical blocks and the typical sort-merge networks. Such networks (e.g., Batcher's) generally have a recursive structure: the sorting network for n elements consists of two such sorting networks for $n/2$ elements applied separately to disjoint halves, followed by a merging network for the two sorted halves. Such a network can be thought of as a sequence of $\lg n$ different networks with the j^{th} network separately merging 2^{n-j} pairs of sorted subvectors of size 2^{j-1} into subvectors of size 2^j.

The balanced sorting network does not have this recursive structure since it consists of $\lg n$ identical blocks. While each successive block has the apparent effect of separately merging pairs of subvectors this is not what happens; the set of elements appearing in one of the 2^{n-j} sorted subvectors after the j^{th} block is not necessarily the same set of elements which appeared in the corresponding subvector of the input.

It is certainly possible to use the balanced merging network recursively to construct a typical 'sort-by-merging' network, but this sacrifices a primary advantage of our network that each successive block is identical. For example, the structure of our network permits a hardware implementation consisting of a single block which sorts by looping the output back through as input $\lg n$ times.

3 LOG n BLOCKS ARE SUFFICIENT TO SORT.

Here we present the proof of our main result and mention some related properties of the balanced-merging network. Let the input size be $n=2^k$. We want to show that k blocks applied in succession sort any input. The basic idea is to show that each successive block decreases the "disorder" of the vector in the sense that successively larger subvectors are sorted until the vector is completely sorted after the k^{th} block.

By the zero-one principle [K] it is sufficient to analyze the network for binary input. However, we do the analysis for general input to give more insight into the operation of the network.

In what follows, it is helpful to represent each number in the index set $\{0,1,\ldots,2^k-1\}$ as a binary string of length k. When we refer to the bits of an index we mean the bits of its binary representation.

The subvectors of x which are relevant to our results are as follows: The even chain (resp. odd

chain) of x is the subvector of entries with even (resp. odd) indices. These are the level one chains of x. In general, the set of level i chains of x, $0 \le i \le k$, consists of 2^i subvectors each of size 2^{k-i} and containing the entries whose indices have the same rightmost i bits.

The A-cochain of x (resp. B-cochain of x) consists of entries corresponding to indices whose rightmost 2 bits are 00 or 11 (resp., 01 or 10). These are the level one cochains of x. By dropping the rightmost bit from the indices of the cochains, we obtain the index set $\{0,1,\ldots,2^{k-1}-1\}$ for each of the A and B cochains. We can then partition the A-cochain (resp. B-cochain) into the AA-cochain and AB-cochain (resp. BA-cochain and BB-cochain) of x in the same way by ignoring the rightmost bit. These are the level 2 cochains. Recursively, we define the level i cochains by partitioning the level $i-1$ cochains in the same way. One shows easily that there are 2^i level i cochains each of 2^{k-i} entries. Each level i cochain is defined by a pair of complementary bit strings $(\alpha, \bar{\alpha})$ of length $i+1$, and consists of the entries of x corresponding to indices whose rightmost $i+1$ bits are α or $\bar{\alpha}$. (See Figure 4)

Our main result, stated precisely is:

Theorem 2: If x is a 2^k-vector then applying j blocks successively to x $(0 \le j \le k)$ yields a vector whose level $k-j$ chains are sorted. In particular applying k blocks yields a sorted vector.

The theorem is proved by induction on j. For $j = 0$ the theorem is trivial. For $0 < j \le k$, we assume that the chains at level $k-(j-1)$ are sorted after the first $j-1$ blocks. We will prove the following three lemmas. (These lemmas are demonstrated in Figure 5 where a vector of 16 elements is sorted by a balanced sorting network.)

Lemma 3: Applying the first j phases of a block to a 2^k-vector whose level $k-(j-1)$ chains are sorted yields a vector whose level $k-j$ cochains are sorted.

Lemma 4: Applying the $(j+1)^{st}$ phase $(j < k)$ to a vector whose level $k-j$ cochains are sorted yields a vector whose level $k-j$ chains are sorted.

Lemma 5: Applying the i^{th} phase $(j+1 < i \le k)$ to an input whose level $k-j$ chains are sorted preserves this property.

The three lemmas together with the induction hypothesis immediately imply that the level $k-j$ chains are sorted after the j^{th} block is applied. (For the k^{th} block, note that lemma 3 alone implies that the output is sorted). This proves the theorem.

It is therefore sufficient to prove these lemmas. This will require some preliminary analysis of the properties of phases and blocks.

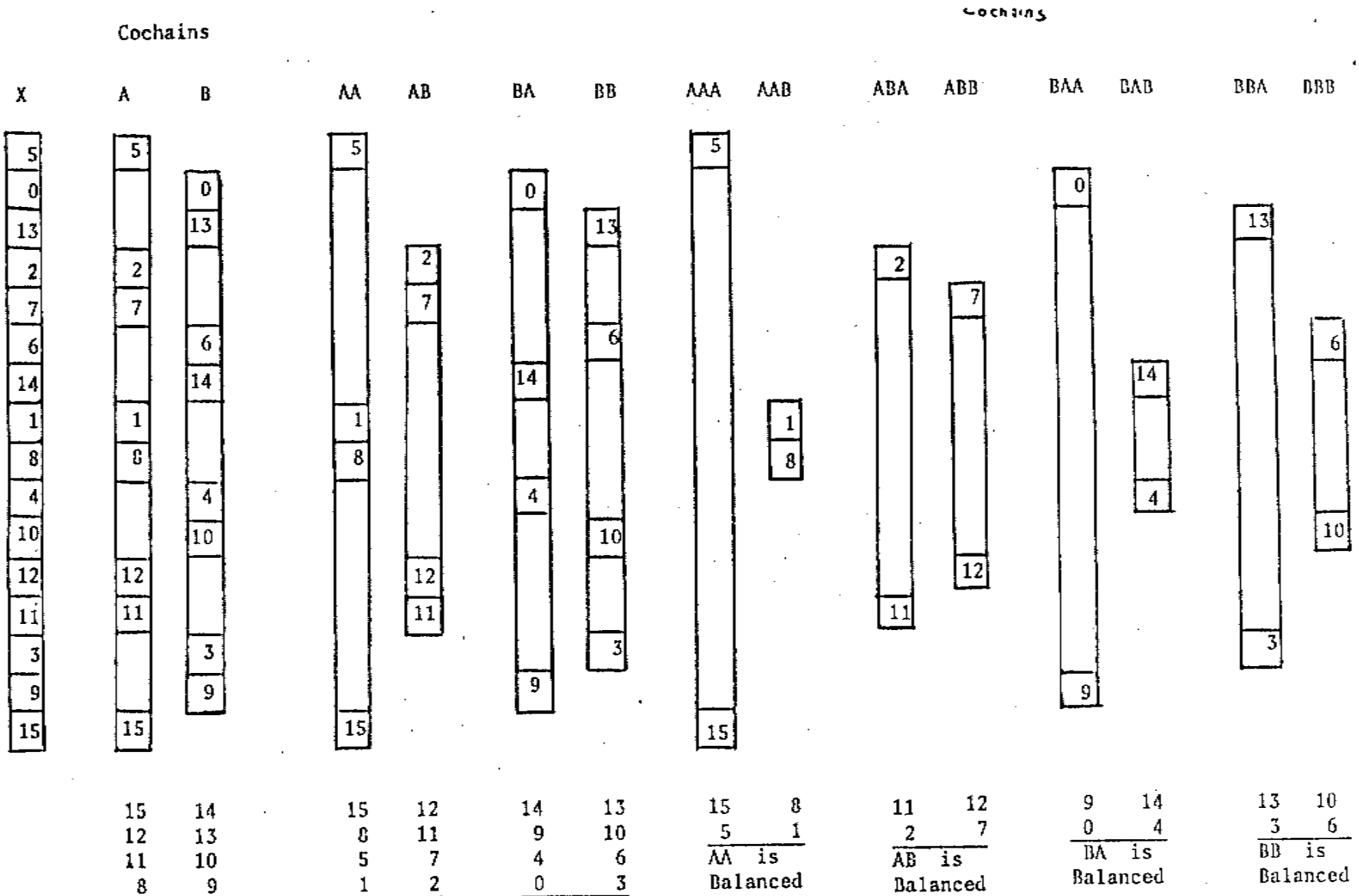


FIGURE 4. The cochains of a 16 element vector X, that is recursively balanced.

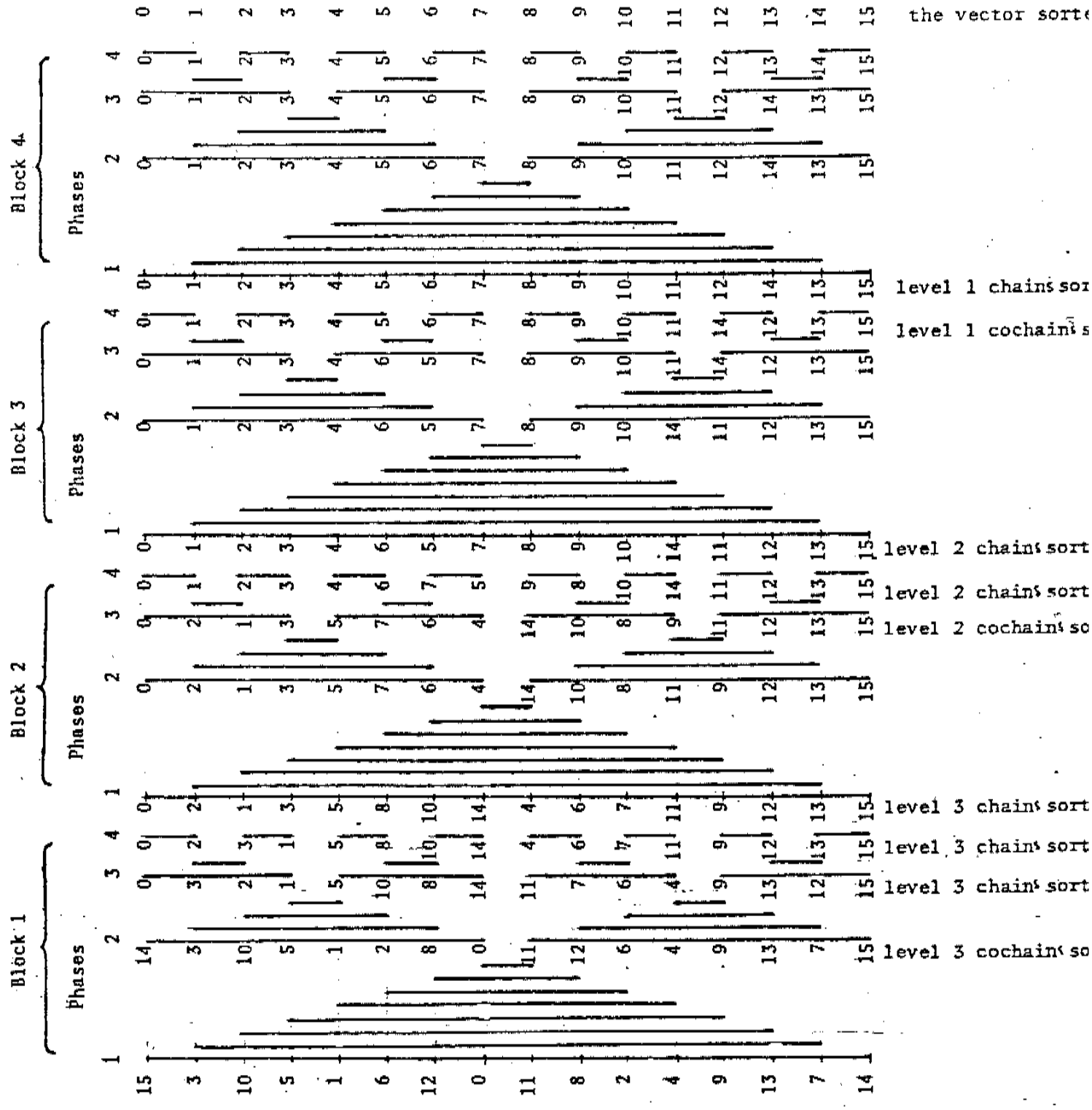


Figure 5.

An Example of Sorting 16 Numbers with the Balanced Network.
(Each Block Yields Longer Sorted Chains and Cochains.)

The definition of the phases of a block immediately implies:

Proposition 6: The j^{th} phase of a 2^k-block compares all pair of entries whose indices have identical leftmost $(j-1)$ bits and complementary rightmost $k-(j-1)$ bits.

We will use the following technical lemma that is proved in [DPRS]:

Lemma 7: Let x be a 2^k-vector and i,j be nonnegative integers with $i+j<k$. Then

i) The level i chains of the level j chains of x are the level $(j+i)$ chains of x.

ii) The level i cochains of the level j cochains of x are the level $(j+i)$ cochains of x.

iii) The level i chains of the level j cochains of x are the level $(j+i)$ chains of x.

Proof: i) A level j chain y of x consists of entries whose indices have the same rightmost j bits. A level i chain of y consists of entries whose indices are equal in bit positions $j+1$ to $j+i$ and is thus a level $i+j$ chain of x.

ii), iii) By induction on i, consider first the case $i=1$. Let y be any level j cochain of x, specified by complementary bit strings $(\alpha, \bar{\alpha})$ of length j + 1 (where the first bit of α is 0). The rightmost j + 2 bits of the indices of y are, alternately 0α, 0α, 1α, $1\bar{\alpha}$. The A-cochain (resp. B-cochain) of y consists of all entries with rightmost j + 2 bits 0α or $1\bar{\alpha}$ (resp. $0\bar{\alpha}$ or 1α) and thus is a level j + 1 cochain of x. Similarly the even (resp. odd) chain of y consists of all entries with rightmost j + 1 bits g[a] (resp. $\bar{\alpha}$) and thus are level j + 1 chains of x.

To prove the induction step for (ii) suppose that $i=h>1$ and that the theorem is true for $i<h$. Let z be a level h cochain of the level j cochain y of x. By the induction hypothesis, z is a level 1-cochain of some level $h-1$ cochain w of y. Again by the induction hypothesis, w is a level $j+h-1$ cochain of x. Applying the induction hypothesis a third time implies that z is a level $(j+h-1)+1=j+h$ cochain of x.

The induction step for (iii) follows directly from the case $i=1$ and (i). □

Next we observe:

Lemma 8: If $j\le r<k$ then applying the j^{th} phase of a 2^k-block to x is equivalent to applying the j^{th} phase of a 2^r-block independently to each level $k-r$ cochain of x.

Proof: Let $m=2^{k-r}$ and let $y_1, y_2, \ldots, y_m$ be the level $k-r$ cochains of x. It suffices to show that each comparison performed when applying the j^{th} phase of a 2^r block to a y_i is performed by the j^{th} phase of a 2^k-block applied to x.

case i ($r = j$): Each y_i is defined by a pair of complementary bit strings (α, $\bar{\alpha}$) of length $k-j+1$. The j^{th} phase of a 2^j block applied to a y_i compares the $2h^{th}$ entry of y_i with the $(2h+1)$ entry of y_i for each $0 \le h \le 2^{j-1}-1$. The indices for the $2h^{th}$ and $(2h+1)^{th}$ entry of y_i have the same leftmost $j-1$ bits and the rightmost bits are α and $\bar{\alpha}$. Thus by proposition 6 each such comparison is done by the j^{th} phase of a 2^k-block applied to x.

case ii ($r<j$): Applying the j^{th} phase of a 2^r-block to y_i is, by case (i), equivalent to applying the j^{th} phase of a 2^j-block to every level $r-j$ cochain of y_i, which by lemma 7 (ii), is a level $(k-j)$-cochain of x. By case (i), these are the same comparisons done in the j^{th} phase of a 2^k-block applied to x. □

We now characterize those vectors that are sorted by a single block.

A vector x is balanced if, when the entries are arranged in nondecreasing order in pairs, one entry from each pair lies in the A-cochain of x and the other in the B-cochain of x. (Equivalently, for each i, the j^{th} largest entry of the A-cochain (resp. B-cochain) of x is less than or equal to the $(i+1)^{th}$ largest entry of the B-cochain (resp. A-cochain). The vector x is recursively balanced if it is balanced and the A-cochain and B-cochain of x are each recursively balanced. (Here any vector with two entries is defined to be recursively balanced). The cochains and a recursively balanced input are depicted in Figure 4.

Lemma 9: The last phase of a block sorts y if and only if y is balanced and both level one cochains of y are sorted.

Proof: The last phase compares $y(2i)$ to $y(2i+1)$ for all $0 \le i \le 2^{k-1}-1$. Thus the output is sorted if and only if we have $y(0), y(1) \le y(2), y(3) \le \ldots \le y(n-2), y(n-1)$ which holds if and only if y is balanced and each level one cochain of y is sorted. □

Lemma 10: A 2^k-vector is sorted by a single block if and only if it is recursively balanced.

Proof: By induction on k, the case $k = 1$ is trivial. By the previous lemma, x is sorted by a single block if and only if the first $k-1$ phases produce a balanced vector y whose level one cochains are sorted. By Lemma 8 the first $k-1$ phases of a 2^k-block act independently on each level one cochain as a 2^{k-1}-block. Thus, y is balanced and by the induction hypothesis, each level one cochain of y is sorted if and only if x is balanced and each level one cochain of x is recursively balanced, i.e. x is recursively balanced. □

We digress momentarily, to determine how many inputs are sorted by a single block, i.e. how many permutations of a set of n distinct numbers are recursively balanced.

Proposition 11: The number of recursively balanced permutations of $n = 2^k$ distinct numbers is $n^{n/2}$.

Let $f(n)$ denote the number of recursively balanced permutations of n distinct elements. A permutation is recursively balanced if and only if it is balanced and the A and B cochains are each recursively balanced. If we order the elements in pairs then the result is balanced if exactly one element from each pair is in the A-cochain. There are $2^{n/2}$ ways the elements of the A-cochain can be selected. Once this is done, there are $f(n/2)$ ways to order the A cochain and $f(n/2)$ ways to order the B cochain so we have:

$$f(n) = 2^{n/2}[f(n/2)]^2; f(1) = 1.$$

It is easy to check that $f(n) = n^{n/2}$ is the solution. □

In comparison, any merging network sorts at least $\binom{n}{n/2} = O(2^n/n^{1/2})$ permutations. In a recent paper [P], it was shown that Bitonic and Odd-Even merging networks do more than just merge - they also sort any input vectors that satisfy certain properties similar to our recursively balanced property. We now continue with the proof of the main theorem.

Lemma 12: If the level one chains of the 2^k-vector $\boldsymbol{y}$ are sorted then $\boldsymbol{y}$ is recursively balanced.

Proof: By induction on k, if $k = 1$ the lemma is trivial. Suppose $k > 1$ and let $\boldsymbol{y}_i$, $i = 0,1,2$ or 3 denote the subvector of $\boldsymbol{y}$ with indices congruent to i mod 4, i.e. the level two chains of $\boldsymbol{y}$. By Lemma 7 the level one chains of $\boldsymbol{y}^A$ and $\boldsymbol{y}^B$ (the level one cochains of $\boldsymbol{y}$) are, respectively, $\boldsymbol{y}_0, \boldsymbol{y}_3$ and $\boldsymbol{y}_1, \boldsymbol{y}_2$. Since the level one chains of $\boldsymbol{y}$ are sorted, $\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2$ and $\boldsymbol{y}_3$ are each sorted. So by the induction hypothesis, $\boldsymbol{y}^A$ and $\boldsymbol{y}^B$ are recursively balanced.

To show $\boldsymbol{y}$ is balanced (and thus recursively balanced), we will show that if we arrange the entries of $\boldsymbol{y}$ in pairs in nondecreasing order then for each pair one entry is in $\boldsymbol{y}^A$ and the other is in $\boldsymbol{y}^B$. The even chain of $\boldsymbol{y}$ is obtained by alternating entries of $\boldsymbol{y}_0$ and $\boldsymbol{y}_2$ and the odd chain is obtained by alternating entries in $\boldsymbol{y}_1$ and $\boldsymbol{y}_3$. Thus both chains consist of entries alternately in $\boldsymbol{y}^A$ and $\boldsymbol{y}^B$ with one of them starting in $\boldsymbol{y}^A$ the other in $\boldsymbol{y}^B$. Since the two chains are each sorted, the two smallest entries of $\boldsymbol{y}$ are either the two smallest entries of the even chain, the two smallest entries of the odd chain, or the smallest entry of each. In any case, the smallest pair consists of one entry in $\boldsymbol{y}^A$ and one in $\boldsymbol{y}^B$. After removing these two elements the remaining elements of each chain are still alternately in $\boldsymbol{y}^A$ and $\boldsymbol{y}^B$ and one of them starts in $\boldsymbol{y}^A$ and the other in $\boldsymbol{y}^B$. Thus the same argument applies for the next pair and each succeeding pair, so $\boldsymbol{y}$ is balanced and recursively balanced. □

Corollary 13: If the even and odd chains of $\boldsymbol{x}$ are each sorted, a single block sorts $\boldsymbol{x}$.

Definition 14: We say that a 2^k-vector $\boldsymbol{x}$ dominates a 2^k-vector $\boldsymbol{y}$ if $\boldsymbol{x}(i) \geq \boldsymbol{y}(i)$ for all i.

Proposition 15: If $\boldsymbol{x}$ dominates $\boldsymbol{y}$ then applying an identical sequence of comparisons to $\boldsymbol{x}$ and $\boldsymbol{y}$ preserves the domination.

Proof: It suffices to show that domination is preserved by a single comparison applied to both x and y. If the i^{th} element is compared to the j^{th} element (where $i<j$) then in the new vectors x' and y' we have

$$x'(i) = min\{x(i), x(j)\} \geq min\{y(i), y(j)\} = y'(i)$$

and

$$x'(j) = max\{x(i), x(j)\} \geq max\{y(i), y(j)\} = y'(j).$$

Thus x' dominates y'. □

We are now ready to prove lemmas 3, 4 and 5.

Proof (of Lemma 3): Let $m = 2^{k-j}$ and let $x_1, x_2, \ldots, x_m$ be the level $k-j$ cochains of x (each having 2^j-entries). By lemma 8, the first j phases of a 2^k-block act independently on each x_i as a 2^j-block. By lemma 7, the level one chains of each x_i are level $k-(j-1)$ chains of x and are sorted, by hypothesis. By corollary 13, applying a 2^j-block to each x_i sorts it. □

Proof (of Lemma 4): Consider first the case $j = k-1$. Then the k^{th} (final) phase of a block is applied to a vector x whose level one cochains are sorted. If y is the output vector then

$$y^E(i) = y(2i) = min\{x(2i), x(2i+1)\} = min\{x^A(i), x^B(i)\}$$
$$y^O(i) = y(2i+1) = max\{x(2i), x(2i+1)\} = max\{x^A(i), x^B(i)\};$$

since for i even $x(2i) = x^A(i)x(2i+1) = x^B(i)$ and for i odd $x(2i) = x^B(i)x(2i+1) = x^A(i)$. Since x^A and x^B are each sorted y^E and y^O are sorted.

If $k-j>1$, let $m = 2^{k-(j+1)}$ and let $x_1, x_2, \ldots, x_m$ be the level $k-(j+1)$ cochains of x (each having 2^{j+1} entries). Then by lemma 8, the $(j+1)^{th}$ phase applied to x is equivalent to applying the last phase of 2^{j+1}-block separately to each level $k-(j+1)$ cochain of x. The level one cochains of each x_i are level $k-j$ cochains of x and are sorted, by hypothesis. By the argument of the previous paragraph, the last phase of a 2^{j+1} block applied to each x_i yields vectors y_i whose level one chains are sorted. The level one chains of the level $k-(j+1)$ cochains are just the level $k-j$ chains of the output y. □

Proof (of Lemma 5) Set m = 2^j and let $x_1, x_2, \ldots, x_m$ denote the subvectors of x with x_1, consisting of the first 2^j entries of x, x_2 consisting of the next 2^j entries of x, etc. The level $k-j$ chains of x are sorted. Thus x_{i+1} dominates x_i. Each phase after the $(j+1)^{th}$ acts separately and identically on each x_i (by the recursive definition of a phase), thus by proposition 15 the domination of x_i by x_{i+1} is preserved. □

This completes the proof of Theorem 1.

By examining the proof of Theorem 1 we see that not every phase of each of the k blocks is required for the network to sort. For the induction to work the output of the j^{th} block has its level $k-j$ chains sorted. By lemmas 3 and 4 only the first $j+1$ phases of the j^{th} block are needed. Thus we have:

Theorem 16: The network consisting of k partial blocks where the j^{th} partial block ($j<k$) consists of the first $j+1$phases and the final partial block consists of all k phases sorts any input.

Finally, a natural question to ask is: what vectors can be the output of a single block. We say a 2^k-vector x satisfies dominance if the vector consisting of the first 2^{k-1} entries is dominated by the vector consisting of the second 2^{k-1} entries, i.e. $x(i)<x(2^{k-1}+i)$, for $0\le i\le 2^{k-1}-1$. The vector y satisfies recursive dominance if it satisfies dominance and its upper half and lower half each satisfy recursive dominance. (E.g., the vector (1, 4, 2, 7, 3, 6, 5, 8) satisfies recursive dominance.) S. A. Cook observed that the output of a block satisfies recursive dominance; in fact this is necessary and sufficient. The proof of the following theorem can be found in [DPRS].

Theorem 17: The n-vector y can be the output of a block if and only if it satisfies recursive dominance.

Proof: First we prove by induction on $n=2^k$ that the output of a block satisfies recursive dominance. Let x by any n-vector and let y be the output of a block applied to x. Let z be the output of the first phase applied to x. Then y is the result of applying the 2^{nd} through k^{th} phases to z. These phases act separately as n/2- blocks on each half of z so by the induction hypothesis, both halves of y satisfy recursive dominance. It remains to show that y satisfies dominance. Note that $z(i)\le z(n-1-i)$ for $0\le i\le n/2-1$ by the definition of the first phase. The vector w resulting from the second phase is given by

$$w(i)=min\{z(i),z(n/2-1-i)\}$$

$$w(n/2+i)=min\{z(n/2+i),z(n-1-i)\} \qquad 0\le i\le n/4-1$$

and

$$w(i)=max\{z(i),z(n/2-1-i)\}$$

$$w(n/2+i)=max\{z(n/2+i),z(n-1-i)\} \qquad n/4\le i\le n/2-1$$

Thus $w(i)\le w(n/2+i)$ for $0\le i\le n/2-1$ and w satisfies dominance. Subsequence phases act identically on the upper and lower half of w and so, by proposition 15, preserve the dominance property. Hence y satisfies recursive dominance.

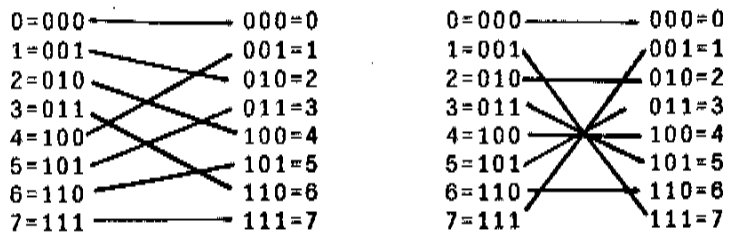
Next we show by induction every n-vector y satisfying recursive dominance can be the output of a block. It is easy to see that the even and odd chain y^E and y^O of y both satisfy recursive dominance. By the induction hypothesis there are n/2-vectors w^E and w^O such that an $n/2$ block applied to w^E (resp. w^O) yields y^E (resp. y^O. Let x be the vector whose A-cochain is w^E and whose B-cochain is w^O. By Lemma 8, the first k-1 phases acts separately as two $n/2$ blocks on the A and B cochains, thus yielding a vector z whose A cochain is y^E and whose B cochain is y^O. Applying the k^{th} phase to z does not move elements with indices congruent to 0 or 1 mod 4 and switches the elements with indices congruent to 2 or 3 mod 4 since y satisfies recursive dominance and thus yielding y. □

4 SHUFFLE EXCHANGE IMPLEMENTATION

In this section we present an implementation of the balanced sorting network using the shuffle exchange interconnection model [St]. We first define the shuffle-exchange connection and then show that the shuffle-based layout performs the same comparisons as does the balanced sort network. The section concludes with some observations about the layout.

4.1 The Shuffle Permutation

A shuffle exchange layout for $n = 2^k$ 'elements' consists of a series of identical stages. Each stage consists of $n/2$ two by two comparators, numbered 0 to $n/2-1$. If we number the lines through the comparators in a stage from 0 to $n-1$ so that the lines through the i^{th} comparator are labeled $2i$ and $2i+1$ then output i from stage t is connected to input σ(i) in stage t + 1 where the permutation σ is the shuffle permutation: if $i_{k-1}\, i_{k-2} \ldots i_0$ is the binary expansion for i then $\sigma(i) = i_{k-2}\, i_{k-3} \ldots i_0\, i_{k-1}$ (see Figure 6(a)).



(a) SHUFFLE PERMUTATION σ (b) THE PERMUTATION τ

FIGURE 6

Each comparator comparing input lines i and j, $i<j$ can be set into four possible states:

1.State + : output(i) $\leq$ output(j) (*larger value to the upper line*)
2.State − : output(i) $\geq$ output(j) (*larger value to the lower line*)
3.State 0 : output(i) = input(i),
output(j) = input(j) (*no exchange*)
4.State 1 : output(i) = input(j),
output(j) = input(i) (*exchange*).

4.2 The Layout

Throughout this section, the word "network" refers to the balanced merging network and the word "layout" refers to the shuffle exchange implementation. The layout which implements the balanced merging network (a single block of the balanced sorting network) consists of $\lg n$ shuffle exchange stages with all comparators set to the " + " state. Each stage corresponds to a phase of the merging network. In order that the layout simulate the merging network, the inputs into the layout must be a certain permutation τ of the inputs into the network, given by $\tau(2i) = 2i$ and $\tau(2i+1) = n-2i-1$, that is, τ fixes the location of the even inputs and reverses the order of the odd inputs (see Figure 6(b)).

We now proceed to show that the output after each phase of the merging network for a given input is precisely the output after each corresponding stage of the above layout with the same input (permuted by τ). In order to show the correspondence between the network and the proposed layout, we introduce some notation.

Definition 18: Let Line^t(i) be the value on the i^{th} network line (see Figure 1) just before the phase t comparisons. In particular, Line^1(i) are the input values.

Definition 19: Let $\underline{\text{In}^t(i)}$ be the value of the i^{th} input line of the t^{th} stage of the layout. This is the value for the i(mod 2)th input into the $\lfloor i/2 \rfloor$ comparator. Note that during the t^{th} stage comparator i compares $\text{In}(2i)$ and $\text{In}(2i+1)$.

We use greek letters to represent a string of bits and superscript the letter to indicate the number of bits. For example $\alpha^{k-j}\,\beta^{j-1}\,1$ indicates a string of k bits, the first (high order) $k-j$ bits of which are denoted by α^{k-j}, and the last bit is 1. The initial relationship between the inputs of the layout and the network is given by:

$$\text{In}^1(2i) = \text{Line}^1(2i) \quad \text{and} \quad \text{In}^1(2i+1) = \text{Line}^1(n-2i-1)$$

Thus the first stage of comparators of the layout performs the same comparisons as the first phase of the sorting network. Figure 7 tracks the movement through the layout over time with the numbers in the figure corresponding to the location of the corresponding input line in the balanced merging network. All comparators are in the state " + ".

According to Proposition 6, the j^{th} phase of the balanced merging network compares those pairs of entries whose indices are identical in the first (rightmost) $j-1$ bits and complementary in the remaining $k-(j-1)$ bits. We want to show that the same comparisons are performed during the j^{th} stage of the layout. Restated in our notation this becomes

Theorem 20: For $1 < j \le k$, and $0 \le i \le n/2$

$$\text{In}^j(2i) = \text{Line}^j(\alpha^{j-1}\,\beta^{k-(j-1)})$$
$$\text{In}^j(2i+1) = \text{Line}^j(\bar{\alpha}^{j-1}\,\beta^{k-(j-1)}).$$

Before proving the theorem, we first present two propositions and a lemma relating the values at stage t with those at stage $t-1$. The first proposition explicitly shows that after a network comparison, the minimum of the two compared values is then placed onto the smaller (horizontal) line number. The second proposition states a similar fact for the layout: the minimum value from the comparison goes to the shuffle connection emanating out of the top (0) output port of the comparator.

Proposition 21: For $1 < j \le k$

$$\text{Line}^j(\alpha^{j-2}0\beta^{k-(j-1)}) = \min\{\text{Line}^{j-1}(\alpha^{j-1}0\beta^{k-(j-2)}), \text{Line}^{j-1}(\alpha^{j-1}1\bar{\beta}^{k-(j-2)})\}$$

$$\text{Line}^j(\alpha^{j-2}1\bar{\beta}^{k-(j-1)}) = \max\{\text{Line}^{j-1}(\alpha^{j-1}0\beta^{k-(j-2)}), \text{Line}^{j-1}(\alpha^{j-1}1\bar{\beta}^{k-(j-2)})\}$$

Proposition 22: For $1 < j \le k$

$\ln^j(\chi^{k-2}0\delta^1) = \min\{\ \ln^{j-1}(\delta^1\chi^{k-2}0),\ \ln^{j-1}(\delta^1\chi^{k-2}1)\ \}$

$\ln^j(\chi^{k-2}1\delta^1) = \max\{\ \ln^{j-1}(\delta^1\chi^{k-2}0),\ \ln^{j-1}(\delta^1\chi^{k-2}1)\ \}$

The following lemma relates the inputs to the comparitors of the layout with those of the network. Its proof follows directly, but tediously, from the previous two propositions and can be found in [DPRS].

Lemma 23:

$\ln^1(\alpha^{k-1}0) = \mathrm{Line}^1(\alpha^{k-1}0)$
$\ln^1(\alpha^{k-1}1) = \mathrm{Line}^1(\bar{\alpha}^{k-1}1)$

and for $j>1$,

$\ln^j(\alpha^{k-j}\beta^{j-2}00) = \mathrm{Line}^j(\beta^{j-2}0\alpha^{k-j}0)$
$\ln^j(\alpha^{k-j}\beta^{j-2}01) = \mathrm{Line}^j(\beta^{j-2}0\bar{\alpha}^{k-j}1)$

$\ln^j(\alpha^{k-j}\beta^{j-2}10) = \mathrm{Line}^j(\beta^{j-2}1\alpha^{k-j}1)$
$\ln^j(\alpha^{k-j}\beta^{j-2}11) = \mathrm{Line}^j(\beta^{j-2}1\alpha^{k-j}0)$

Proof: For $j=1$ the proof follows immediately from the initialization. For $j>1$, we assume the lemma holds for $j-1$. By Proposition 22,

$\ln^j(\alpha^{k-j}\beta^{j-2}00) = \min\{\ \ln^{j-1}(0\alpha^{k-j}\beta^{j-2}0),\ \ln^{j-1}(0\alpha^{k-j}\beta^{j-2}1)\}$
Substituting $\mu^{n-(j-1)}$ for $0\alpha^{n-j}$
and $\nu^{j-3}\delta^1$ for β^{j-2}

$= \min\{\ln^{j-1}(\mu^{k-(j-1)}\nu^{j-3}\delta^10),\ \ln^{j-1}(0\mu^{k-(j-1)})\nu^{j-3}\delta^11)\ \}$

Case $\delta^1 = 0$:

$= \min\{\ \ln^{j-1}(\mu^{k-(j-1)}\nu^{j-3}00),\ \ln^{j-1}(0\mu^{k-(j-1)}\nu^{j-3}01)\ \}$
By the induction hypothesis.

$= \min\{\ \mathrm{Line}^{j-1}(\nu^{j-3}0\mu^{k-(j-1)}0),\ \mathrm{Line}^{j-1}(\nu^{j-3}0\bar{\mu}^{k-(j-1)}1)\ \}$
Substitute back μ and ν.

$= \min\{\ \mathrm{Line}^{j-1}(\beta^{j-2}0\alpha^{k-j}0),\ \mathrm{Line}^{j-1}(\beta^{j-2}1\bar{\alpha}^{k-j}1)\ \}$
By Proposition 21.

$= \text{Line}^j(\beta^{j-2}0\alpha^{k-j}0)$

Case $\delta^1 = 1$:

$= \min\{\text{In}^{j-1}(\mu^{k-(j-1)}\nu^{j-3}10), \text{In}^{j-1}(0\mu^{k-(j-1)}\nu^{j-3}11)\}$

$= \min\{\text{Line}^{j-1}(\nu^{j-3}1\bar{\mu}^{k-(j-1)}1), \text{Line}^{j-1}(\nu^{j-3}1\mu^{k-(j-1)}0\}$

$= \min\{\text{line}^{j-1}(\beta^{j-2}1\bar{\alpha}^{k-j}1), \text{Line}^{j-1}(\beta^{j-2}0\alpha^{k-j}0)\}$

$= \text{Line}^j(\beta^{j-2}0\alpha^{k-j}0)$

The other cases are similarly proved. □

We are now ready to prove the theorem stating that the layout performs the same comparisons as does the balanced merging network.

Proof: (of Theorem 20): Let λ^{k-1} be the binary representation of i.

Case 1: λ^{k-1} is of the form $\alpha^{k-j}\beta^{j-2}0$. Then by the previous lemma,

$$\text{In}^j(2i) = \text{In}^j(\lambda^{k-1}0) = \text{In}^j(\alpha^{k-j}\beta^{j-2}00)$$
$$= \text{Line}^j(\beta^{j-2}0\alpha^{k-j}0) = \text{Line}^j(\beta^{j-1}\alpha^{k-(j-1)})$$

$$\text{In}^j(2i+1) = \text{In}^j(\lambda^{k-1}1) = \text{In}^j(\alpha^{k-j}\beta^{j-2}01)$$
$$= \text{Line}^j(\beta^{j-2}0\bar{\alpha}^{k-j}1) = \text{Line}^j(\beta^{j-1}\bar{\alpha}^{k-(j-1)})$$

Case 2: λ^{k-1} is of the form $\alpha^{k-j}\beta^{j-2}1$. Then by the previous lemma,

$$\text{In}^j(2i) = \text{In}^j(\lambda^{k-1}0) = \text{In}^j(\alpha^{k-j}\beta^{j-2}10)$$
$$= \text{Line}^j(\beta^{j-2}1\bar{\alpha}^{k-j}1) = \text{Line}^j(\beta^{j-1}\alpha^{k-(j-1)})$$

$$\text{In}^j(2i+1) = \text{In}^j(\lambda^{k-1}1) = \text{In}^j(\alpha^{k-j}\beta^{j-2}11)$$
$$= \text{Line}^j(\beta^{j-2}1\alpha^{k-j}0) = \text{Line}^j(\beta^{j-1}\alpha^{k-(j-1)}).$$ □

4.3 The Layout for Sorting

We are now in a position to discuss the layout realizing the balanced sorting network in more detail. Three alternatives are presented, although, many variations and combinations of the schemes are possible.

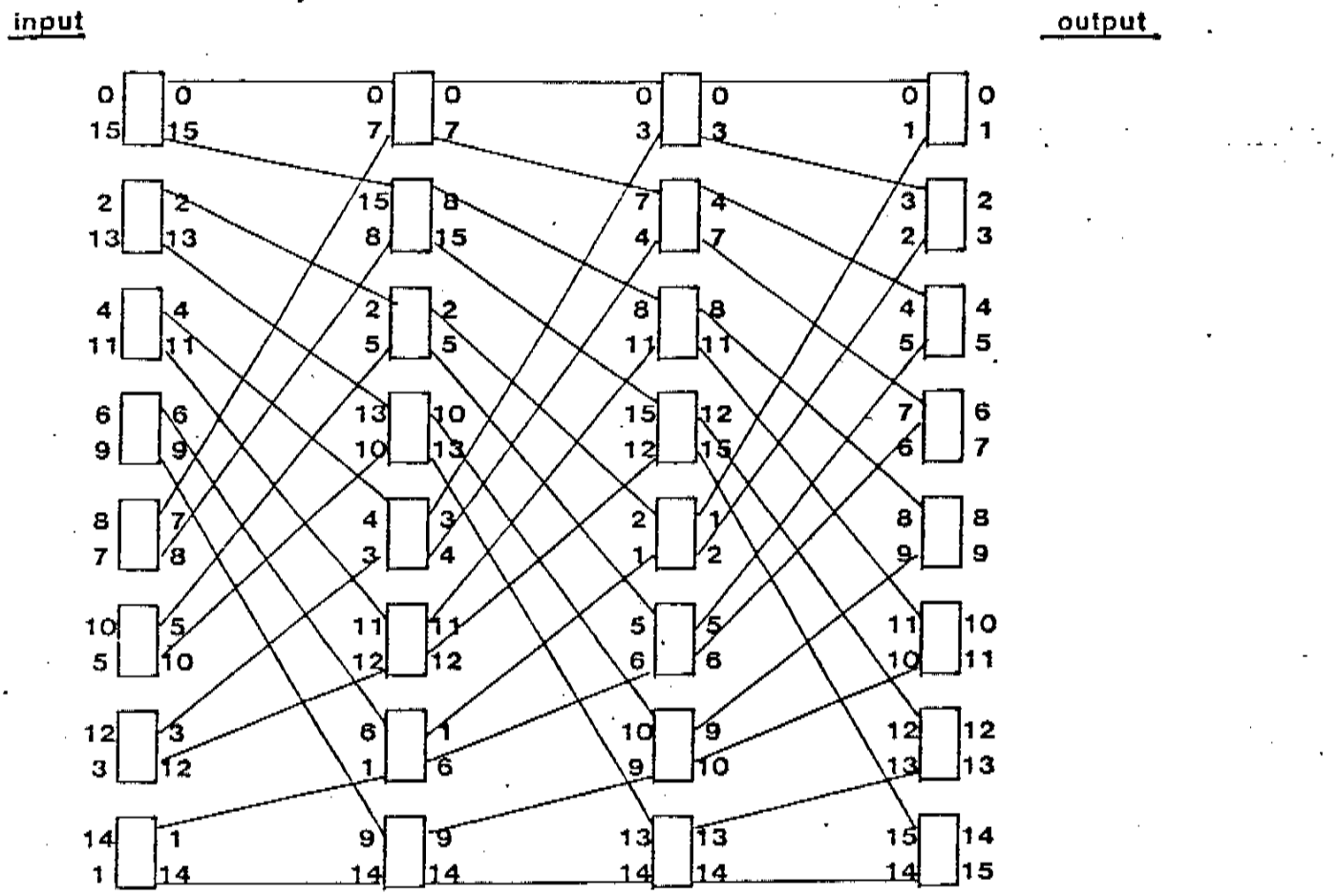


Figure 7.

One block of the layout for 16 items. The numbers on the input and output ports of the comparitors indicate the corresponding horizontal network lines compared in Figure 1. (The result of a comparison places the smaller value onto the lower numbered network line.)

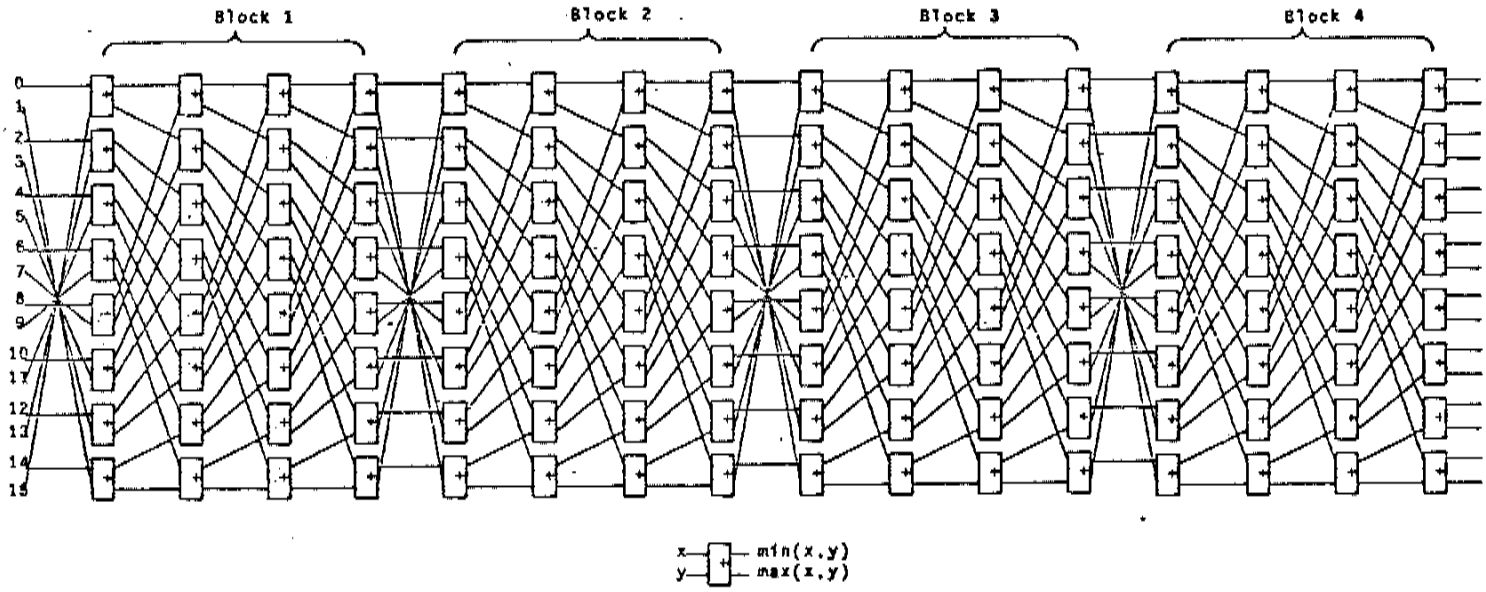


FIGURE 8

The suffle-exchange layout for the balanced sorting network for 16 elements.

Theorem 20 shows that the values of the output lines of the merging layout are the same as those of the merging network. Thus to realize the balanced sorting network we need to only connect together k copies of the layout with the inputs permuted by τ. This is depicted in Figure 8. Note that all comparators are in the " + " state.

Since successive blocks are identical the following variant is possible. One block is layed out as above ($\lg n$ stages, each connected by σ) with the block output recirculated back into the block input according to the permutation τ. Instead of keeping track of $\lg n$ recirculations through the block to produce a sorted output, we appeal to Lemma 1 which states that the output of a block is sorted iff no comparison performs a swap. Thus, the AND of all the actions of the comparators in a block is sufficient to control the direction of the block output (i.e. to recirculate or not). In [R] this scheme is slightly modified to produce a 'robust' VLSI chip or wafer sorting network with the property that any comparitor found to be faulty can be bypassed without affecting the sorting (only the sorting time). In fact, on the average, many comparitors can be bypassed.

A final alternative is to implement the connection permutation τ by $\lg n$ stages of the shuffle exchange permutation using only the 0 and 1 states. The resulting implementation would then consist of one stage of $n/2$ comparitors, with the output recirculated back as input. Sorting would be accomplished in 2 $\lg^2 n$ recirculations. (See [Sc] for a description of a parallel machine, based on shuffle exchange, for which this method is well suited.)

5 REFERENCES

[AKS] M. Ajtai, J. Komlos and E. Szemeredi, "An $O(n \log n)$ sorting network", Proceedings of the 15-th Annual ACM Symposium on Theory of Computing, Boston, 1983, 1-9.

[Ba] K. Batcher, "Sorting networks and their applications," AFIPS Spring Joint Comp. Conf. 32 (1968) 307-314.

[Be] R. Becker, "Sorting out sorting," a film produced in the Dept. Computer Science, University of Toronto, Canada, 1981.

[DPRS] M. Dowd, Y. Perl, L. Rudolph, and M. Saks, "The Balanced Sorting Network," DCS-TR-127, Dept. Computer Science, Rutgers University, New Brunswick, NJ, 1983.

[Ho] C.A.R. Hoare, "Quicksort" Computer J. 5 (1962) 10-15.

[HS] Z. Hong and R. Sedgewick, "Notes on Merging Networks," Proceedings of the 14-th Annual ACM Symposium on Theory of Computing, San Francisco 1982, 296-302.

[K] D.E. Knuth, The art of computer programming, Vol. 3: Sorting and searching, Addison Wesley, Reading, MA, 1973.

[P] Y. Perl, "Bitonic and Odd-Even Networks are More Than Merging," DCS-TR-123, Rutgers University, New Brunswick, NJ, 1983.

[RV] J.H. Reif and L.G. Valiant, "A logarithmic time sort for linear size networks", Proceedings of the 15^{th} Annual ACM Symposium on Theory of Computing, Boston 1983, 10-16.

[R] L. Rudolph, "A Robust Sorting Network," Computer Science Tech. Report, Carnegie-Mellon University, Pittsburgh, Pa., 1983.

[Sc] J.T. Schwartz, "Ultracomputers," ACM Transaction on Programming Languages and Systems, 2(1980) 484-521.

[Sh] D.L. Shell, "A high speed sorting procedure," CACM 2 (1959) 30-32.

[St] H.S. Stone, "Parallel processing with the perfect shuffle," IEEE Transaction on Computing, C-20,2 (1971)