

# A MINIMUM AREA VLSI NETWORK FOR $O(\text{LOGN})$ TIME SORTING

## EXTENDED ABSTRACT

G. Bilardi and F. P. Preparata  
Coordinated Science Laboratory  
University of Illinois at Urbana-Champaign  
Urbana, Illinois 61801 USA

### Summary

A generalization of a known class of parallel sorting algorithms is presented, together with a new interconnection to execute them. A VLSI implementation is also proposed, and its area-time performance is discussed. It is shown that an algorithm in the class is executable in  $O(\text{logn})$  time by a chip occupying  $O(n^2)$  area. The design is a typical instance of a "hybrid architecture", resulting from the combination of well-known VLSI networks as the orthogonal trees and the cube-connected-cycles; it also provably meets the  $AT^2 = \Omega(n^2 \log^2 n)$  lower bound for sorters of  $n$  words of length  $(1+\epsilon)\text{logn}$  ( $\epsilon > 0$ ).

### 1. Introduction

Any VLSI sorter of  $n$  terms, with wordlength  $q = (1+\epsilon)\text{logn}$ ,  $\epsilon > 0$ , must satisfy the relationship  $AT^2 = \Omega(n^2 \log^2 n)$ , since it must support a flow of  $\Phi = \Omega(n \text{logn})$  bits through a suitable bisection, and  $AT^2 = \Omega(\Phi^2)$ . This lower bound holds in a suitable VLSI model of computation whose basic assumptions are that the chip is synchronous (transmission time is independent of wire length) and semellective-unilocal (input data are read only once, at pre-specified input ports). Originally proved by Thompson [2] under the word-local restriction [1] for the input format (all the bits of the same word enter the circuit at the same point), the lower bound has recently been extended to nonword-local designs by F. T. Leighton [14].

In a previous paper [3] we have shown that  $AT^2$  optimal VLSI sorters can indeed be constructed for all computations times  $T \in [\Omega(\log^3 n), O(\sqrt{n \text{logn}})]$ .

In this paper we concentrate on "very fast" sorting, i.e., the class of VLSI sorting

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

algorithms whose running time is  $T = \theta(\text{logn})$ . So far only one VLSI design is known to achieve  $\theta(\text{logn})$  computation time: it is based on the orthogonal-trees architecture [5],[6] and implements an algorithm due to Muller and Preparata [7].<sup>(1)</sup> The optimal layout of the orthogonal trees has area  $A = O(n^2 \log^2 n)$  [6], while the lower bound yields  $A = \Omega(n^2)$  for  $T = O(\text{logn})$ . On the other hand, a closer analysis of the algorithm shows that the information flow  $\Phi$  is  $O(n \text{logn})$ , so that the gap between upper and lower bounds is not due to a gap between actual flow and a flow-based lower bound, but rather to the fact that the length of the layout bisection of the orthogonal trees is  $O(\text{logn})$  times as large as the graph bisection.

We will show in this paper that not only the lower bound on the flow, but also the one on the  $AT^2$  measure is tight, by exhibiting a new network capable of sorting in  $A = O(n^2)$  and time  $T = O(\text{logn})$ .

The rather complex network is a typical instance of "hybrid architecture", resulting from the careful interplay of more standard VLSI networks, as the cube-connected-cycles machine, the mesh-connected machine, and the binary-tree machine. The implemented algorithm is of the type first introduced by Preparata [8], although the recursion strategy has been modified to optimize the network area.

We also introduce a new hybrid architecture, called the mesh-of-CCCs, which has very interesting computational features. The cascade of one  $O(\text{logn})$  sorter and one mesh-of-CCCs of proper size will allow us to construct an  $AT^2$ -optimal sorter for any computation time  $T \in [\Omega(\text{logn}), O(\log^3 n)]$ .

Thus we are able to conclude that optimal  $AT^2 = \theta(n^2 \log^2 n)$  sorting is achievable in the

<sup>(1)</sup> Subsequent to the research leading to this paper, we learned of the construction of Aitai, Komlos, and Szemerédi [13], which also achieves  $\theta(\text{logn})$  time; in addition we have devised a VLSI implementation of their scheme also achieving  $AT^2 = \theta(n^2 \log^2 n)$  [15].

entire "meaningful" range of computation times  $T \in [\Omega(\log n), O(\sqrt{n \log n})]$ . (Simple fan-in arguments show that  $\Omega(\log n)$  is a lower bound for the computation time, and  $A = \Omega(n \log n)$  is a consequence of the semellecutive assumption [14], so that computation times slower than  $\theta(\sqrt{n \log n})$  cannot result in smaller area.)

## 2. A Class of Parallel Sorting Algorithms

Several sorting algorithms can be viewed as particular cases of a rather general scheme, which we now describe.

We call COMBINATION the operation that produces from  $m$  sorted sequences, of  $t$  elements each, one sorted sequence of  $mt$  elements. A network implementing this operation is called an  $(m,t)$ -COMBINER. When  $m = 2$ , COMBINATION reduces to conventional merging.

A parallel algorithm for the  $(m,t)$ -COMBINER has been introduced in [8], and is based on the following idea. (2) The  $m$  input sequences  $S_0, \dots, S_{m-1}$  are pairwise merged to compute for each  $i, j \in \{0, 1, \dots, m-1\}$ , and each  $l \in \{0, 1, \dots, t-1\}$ , the number  $C_{ij}(\ell)$  of elements of sequence  $S_j$  that are less than the  $\ell$ -th element of sequence  $S_i$ . The integer  $C_{ij}(\ell)$  is readily obtained as the difference of the ranks of this element in the merge of  $S_i$  and  $S_j$  and in  $S_i$ . By summing the  $C_{ij}(\ell)$ 's over  $j$  we then obtain the rank of the  $\ell$ -th element of  $S_i$  in the output sequence of the COMBINER: thus, to complete the operation, we simply need to store each element in the position specified by its rank. The primitive operation of the scheme -- the merging of two sequences -- can be done, for example, by means of Batchier's bitonic merger [4].

Given  $n = m_1 m_2 \dots m_d$  elements, we can sort them in  $d$  stages according to the following scheme that we call COMBINE-SORT.

At stage 1 we perform  $n/m_1$  combination operations, each on  $m_1$  sequences of 1 element each. At stage 2 we perform  $n/m_1 m_2$  combinations, each on  $m_2$  sequences of  $m_1$  elements each, and at stage  $i$  we perform  $n/m_1 \dots m_i$  combinations, each on  $m_i$  sequences of length  $m_1 \dots m_{i-1}$ . Finally, at stage  $d$  we combine  $m_d$  sequences of length  $n/m_d$  into one sequence of length  $n$ , which is the output of the COMBINE-SORT scheme.

A diagrammatic illustration of the scheme is given in Figure 1 in the form of a rooted tree. Each node of this tree is a suitable combiner. An

(2) A COMBINATION algorithm based on  $k$ -way merging has been recently proposed by F. T. Leighton [14], for several interesting applications, including optimal VLSI sorting.

$(m_i, t_{i-1})$ -COMBINER,  $1 \leq i \leq d$ , performs the combination of  $m_i$  (sorted) sequences of length  $t_{i-1}$ ; here  $t_0 = 1$  and  $t_{i-1} \stackrel{\Delta}{=} m_1 m_2 \dots m_{i-1}$  for  $i > 1$ . Note that each level of the tree corresponds to a stage of the combination scheme, and that there are  $n_i = n/t_i$  nodes at level  $i$ ,  $1 \leq i \leq d$ .

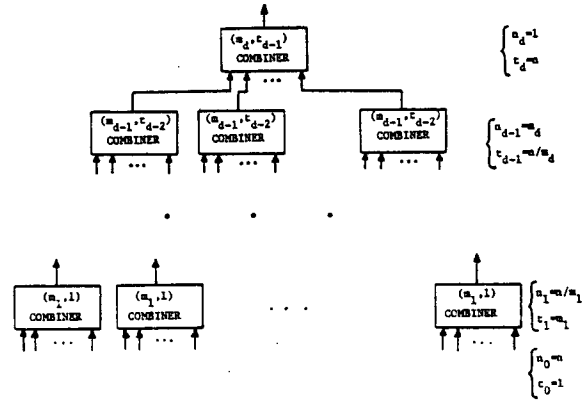


Figure 1. Diagram of the COMBINE-SORT scheme.

Several known sorting algorithms, such as MERGE-SORT and various forms of ENUMERATION SORT [7,8], can be cast in the COMBINE-SORT scheme. Each algorithm is characterized by a particular factorization of  $n = m_1 \dots m_d$  (note that the order of the factors is relevant here), and by the specification of how the combination is to be performed. In this paper we shall explore new significant choices of  $d$  and  $m_1, m_2, \dots, m_d$  that minimize the complexity of a VLSI implementation of COMBINE-SORT.

## 3. An $(m,t)$ -COMBINER Network

In this section we propose a parallel network for an  $(m,t)$ -COMBINER, where  $m = 2^u$  and  $t = 2^v$  are powers of two. This network will accept as input  $m$  sorted sequences of  $t$  elements each,

$$S_i = (s_i(0), s_i(1), \dots, s_i(t-1)) \quad i = 0, 1, \dots, m-1$$

and produce as output a single sorted sequence  $S$ , which is the combination of  $S_0, \dots, S_{m-1}$ , and has  $N = mt \stackrel{\Delta}{=} 2^v$  elements,

$$S = (s(0), s(1), \dots, s(N-1)).$$

The  $(m,t)$ -COMBINER will execute the algorithm based on pairwise merging as outlined in the preceding section. It consists of  $m^2$  modules (each capable of merging two sequences of length  $t$  and of computing partial ranks), laid out as a square  $m \times m$  mesh and indexed as  $M_{ij}$  ( $i, j = 0, 1, \dots, m-1$ ).

The modules of each row are interconnected as the leaves of a binary tree of bandwidth  $t$ ; so are the modules of each column. Thus, the combiner has the structure of an orthogonal-trees machine [5,6], whose leaves are merging modules. As is typical of the orthogonal-trees machine used for sorting, the interconnecting trees broadcast a sequence to all leaves, compute global ranks from partial ranks, and route the elements according to their ranks.

We shall now describe in some detail the merging modules and the interconnecting trees.

3.1. Merging Modules. Merging module  $M_{ij}$  merges sequences  $S_i$  and  $S_j$  and computes  $C_{ij}(\ell)$  for  $\ell = 0, \dots, t-1$ .

Each module is realized (Figure 2) as a cube-connected-cycle (CCC) interconnection of smaller processing elements, called micromodules (each micromodule has a bandwidth of 1 bit). Specifically,

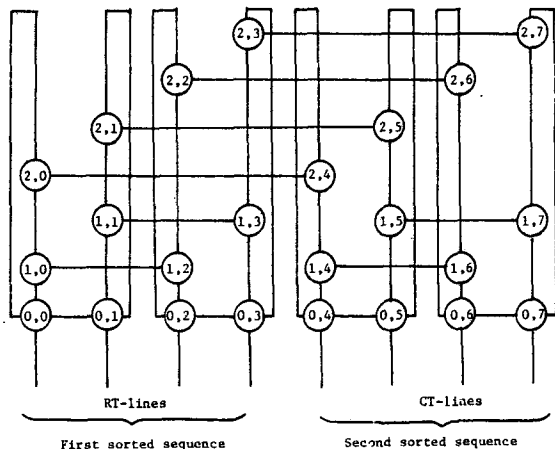


Figure 2. Merging unit  $M_{ij}$  realized by a  $(3, 2^3)$ -CCC, used to merge two sequences with four elements each.

the merging module is a  $(\tau+1, 2^{\tau+1})$ -CCC (i.e., it has  $2^{\tau+1}$  cycles each of length  $\tau+1$ ). We number the micromodules of  $M_{ij}$  as  $M_{ij}(h, k)$ , with  $0 \leq h < \tau+1$  and  $0 \leq k < 2^{\tau+1}$ , so that the merging module may be thought of as a  $(\tau+1) \times 2^{\tau+1}$  array (rows are numbered from bottom to top, columns from left to right). The columns of this array are connected as cycles with a link between  $M_{ij}(h, k)$  and  $M_{ij}(h, (k+1) \bmod (\tau+1))$ . The rows  $0, 1, \dots$ , are respectively associated with the dimensions  $E_0, E_1, \dots, E_\tau$  of a  $(\tau+1)$ -dimensional binary cube [9], and there is a link between  $M_{ij}(h, k_1)$  and  $M_{ij}(h, k_2)$  if and only if the binary expansions of  $k_1$  and  $k_2$  differ exactly in the coefficient of  $2^h$ .

It is well known [10] that the CCC is an efficient emulator of the binary cube interconnection, and the latter is naturally suited to execute a wide class of algorithms, including bitonic sorting. The reader is referred to [10] for a detailed explanation; he must be warned, however, that in the present application the CCC is not used at its full capability, since we deploy a network with  $2t(\log 2t)$  (rather than  $2t$ ) micromodules to merge two sequences of length  $t$ . In other words, a  $2^{\tau+1}$  binary cube is emulated by a  $(\tau+1, 2^{\tau+1})$ -CCC. This is done by reducing the bandwidth of the CCC connection from  $q$  (the word-length) to 1, thereby achieving bit-serial transmission. Thus the pipelining through the modules of a given dimension concerns the different bits of a word, rather than different words of a sequence. (Note that  $\tau+1$  need not be a power of 2. See Figure 2.) Recalling from [10] that a CCC with  $N$  processing elements of constant area can be laid out in area  $O(N^2/\log^2 N)$ , we conclude that a merging module can be laid out in area  $O(A_0 t^2)$ , where  $A_0$  is the constant area of the micromodule.

3.2. Interconnecting Trees. As indicated earlier, the merging modules are interconnected by two families of  $N = mt$  full binary trees, each with  $m = 2^h$  leaves and bandwidth 1. We will refer to these families as the row trees and column trees.

The lines of the row trees and the column trees are respectively labelled  $RT_i(\ell)$  and  $CT_i(\ell)$ ,  $i = 0, \dots, m-1$ ;  $\ell = 0, \dots, t-1$ . The leaves of  $RT_i(\ell)$  are, from left to right, connected to the CCC micromodules  $M_{i0}(0, \ell), M_{i1}(0, \ell), \dots, M_{i, m-1}(0, \ell)$ ; the leaves of  $CT_j(\ell)$  are connected to the CCC micromodules  $M_{0j}(0, t-1+\ell), M_{1j}(0, t-1+\ell), \dots, M_{m-1, j}(0, t-1+\ell)$ .

#### 4. The COMBINATION Algorithm

We now describe how the sorting algorithm of [8], based on pairwise merging, can be executed on the network introduced earlier. This analysis will elucidate the structure of the CCC micromodules.

We recall that the inputs are  $m = 2^h$  sorted sequences of  $t = 2^\tau$  elements each, with  $N = 2^v = mt$ . For convenience we split the algorithm into several phases.

(A) Input of Data and Broadcasting to Merging Modules. Element  $s_i(\ell)$  is input at the root of tree  $RT_i(\ell)$ , and is then broadcast to all leaves of the tree. At this point, the left half of row(0) in module  $M_{ij}$  contains the sequence  $S_i$ . To fill the right halves of row(0) of all modules, we proceed as follows. First, in each "diagonal" module  $M_{ii}$  the sequence  $S_i$  is copied in the second half of row(0). (This can be done by using the connection of row( $\tau$ ) between the left and the right half of the machine.) Next, from micromodule  $M_{jj}(0, t-1+\ell)$ , which is a leaf of  $CT_j(\ell)$ , element

$s_j(\ell)$  is broadcast (through the root) to all the other leaves of the same tree. At this point, the merging module  $M_{ij}$  contains  $S_i$  and  $S_j$  in the 0-th row and merging can begin.

(B) Merging and Rank Computation. Merging can be executed by resorting to the bitonic algorithm, which complies with the DESCEND paradigm of the binary cube (see [10]). It is not difficult to program the CCC to emulate a pipelined merging network (with serial comparison) in time of  $O(\tau+q)$ , where  $q$  is the length of the input words. At the end of merging, the sorted sequence resides in row(0) of the CCC, i.e., the element in  $M_{ij}(0,\ell)$ ,  $0 \leq \ell \leq 2t-1$ , has rank  $\ell$  in  $MERGE(S_i, S_j)$ . To transmit the ranks of  $s_1(0), \dots, s_1(t-1)$  to processors  $M_{ij}(0,0), \dots, M_{ij}(0,t-1)$ , respectively, the path traversed by each element  $s_1(j)$  is retraced backwards: This is easily done with an ASCEND algorithm if each  $M_{ij}(\ell,k)$  keeps track of whether it exchanged or not the operands during the merging process. At the end of this phase, processor  $M_{ij}(0,\ell)$ ,  $0 \leq \ell \leq t-1$ , stores the number of elements in  $MERGE(S_i, S_j)$  that are less than  $s_1(\ell)$ . If from this number we subtract  $\ell$  we obtain  $C_{ij}(\ell)$ , number of elements of  $S_j$  which are less than  $s_1(\ell)$ . We call the  $C_{ij}$ 's partial ranks (stored at the leaves of row trees).

The row trees can now be used in a straightforward manner [6] to compute the total ranks  $C_1(\ell)$  of  $s_1(\ell)$  in time  $O(\mu+\tau)$ , where  $\mu = \log m$  is the depth of the tree and  $(\tau+1)$  is the number of bits of the ranks.

(C) Output of the Sorted Sequence. Our objective is to output element  $s(j2^\tau + \ell)$  from the root of tree  $CT_j(\ell)$ . Consider element  $s_i(h)$  with rank  $C_i(h)$ ,  $h = j2^\tau + \ell$ . The binary spellings of the integers  $j$  and  $\ell$  are the  $\mu$  most significant bits and the  $\tau$  least significant bits of  $C_i(h)$ , respectively. Thus, as a first step, we "activate" in  $M_{ij}$  the elements of sequence  $S_i$  that have to emerge from trees  $CT_j$ 's, and "inhibit" all other elements. (The active elements are those whose rank  $C_i(h)$  agrees in the  $\mu$  most significant bits with the column number  $j$  of the merging module.) Next, we rearrange the active elements in  $M_{ij}$  so that  $s_i(h)$  is sent to  $M_{ij}(0,\ell)$ , with  $\ell = C_i(h) \bmod t$ .

This operation is essentially a permutation of the active (and nonactive) elements, and can be done by using the CCC as an emulator of the Benes network. The setting of the switches, although nontrivial, is greatly simplified with respect to the general case by the fact that the active elements do not change their relative order. The desired rearrangement can be done by using the ideas of concentration introduced in [11], and

expansion, which could be viewed as the inverse of concentration. If  $k$  elements are active in a given module, they are first sent to the  $k$  leftmost columns of the CCC (concentration), and then routed to the destination columns (expansion). A straightforward adaptation of the algorithm that is proposed in [11] for concentration in the cube-machine shows that an ASCEND and a DESCEND phase is all that is required to rearrange data on our CCC. Some bits required to set the switches must be precomputed. This task could be performed by the CCC, or (to keep the micromodule structure as simple as possible), the task can be assigned to a binary tree of full adders whose leaves would be contained in the interface between the CCC and the row trees.

During the entire rearrangement task, computation takes place only in the left-half of the CCC without using dimension  $E_\tau$ . We then transfer each active element from  $M_{ij}(0,\ell)$  to  $M_{ij}(0,t-1+\ell)$ , with a straightforward use of dimension  $E_\tau$ .

At this point element  $s(j2^\tau + \ell)$  is in  $M_{ij}(0,t-1+\ell)$ , (where the value of  $i$  is determined by the input sequence to which  $s(j2^\tau + \ell)$  originally belongs), and is ready to be transmitted to the root of  $CT_j(\ell)$  where it is output.

4.1. Performance Analysis and Modification of the Network. Since both the CCCs and the interconnecting trees work in pipeline in bit-serial mode, any operation takes time proportional to the sum of the operand length and the pipe depth. For the CCC, the depth is  $\tau+1$  and the operand length is either  $q$  (input words) or  $\tau+1$  (partial ranks). Since a constant number of ASCEND and DESCEND algorithms are executed, we conclude that  $O(\tau+q)$  total time is spent in the CCCs. For the trees the depth is  $\mu+1$ , and the operand length is either  $q$  (input words) or  $\tau+\mu$  (total ranks). Since a constant number of fan-in and fan-out algorithms are executed, we conclude that  $O(\tau+\mu+q)$  total time is spent in the trees. Thus the time spent in the interconnecting trees dominates that spent in the CCCs. Recalling that a full binary tree on  $m$  aligned leaves is laid out in area  $\theta(m \log m)$  and that there are  $t$  row and column trees, we conclude

Lemma 1. A full-tree  $(2^\mu, 2^\tau)$ -COMBINER of words of length  $q$  can be laid out in a square of width  $O(\mu 2^{(\tau+\mu)})$  and operates in time  $T = O(\tau+\mu+q)$ .

We now observe that when  $q = \Omega(2^\mu)$ , then  $T = O(\tau+q)$ . In this case the time performance of the trees is insignificantly degraded if we realize them as comb-trees, rather than as full binary trees. The depth increases from  $\mu$  to  $2^\mu$  (which is tolerable in time since  $2^\mu = O(q)$ ), but the layout area decreases by a factor  $O(\mu^2)$ . We conclude:

Lemma 2. A comb-tree  $(2^\mu, 2^\tau)$ -COMBINER of words of length  $q = \Omega(2^\mu)$  can be laid out in a square of width  $O(2^{(\tau+\mu)})$  and operates in time  $T = O(\tau+q)$ .

## 5. A Network for COMBINATION-SORT

The COMBINER can be used to construct a general network for COMBINATION-SORT. As an intermediate step in the construction, we introduce a new operation called COALESCENCE. Given a collection of  $n$  elements, partitioned into  $n/t_{i-1}$  sorted subsequences each containing  $t_{i-1}$  elements, and given a multiple  $t_i$  of  $t_{i-1}$ , which is also a divisor of  $n$ , we call  $(n; t_{i-1}; t_i)$  COALESCENCE the operation of combining (in the sense defined earlier) consecutive blocks of  $m_i = t_i/t_{i-1}$  subsequences.

If we refer to the tree of Figure 1, we can easily see that each level of the tree corresponds to a coalescence of the input sequence. If we call COALESCER a network that performs a coalescence, we can build a COMBINATION-SORTER by cascading a suitable set of coalescers, as shown in Figure 3.

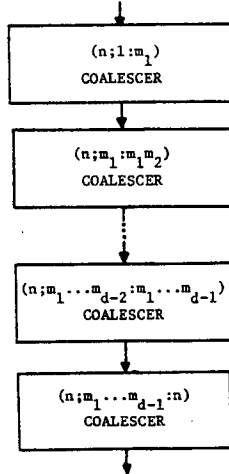


Figure 3. COMBINATION-SORTER as a cascade of COALESCERS.

5.1. The COALESCER. An  $(n; t_{i-1}; t_i)$ -COALESCER can be easily constructed by using  $n_i = n/t_i$   $(m_i, t_{i-1})$ -COMBINERS. Let us assume, for simplicity, that  $n_i$  is a perfect square. We can then lay out the combiners in a  $\sqrt{n_i} \times \sqrt{n_i}$  array with input and output lines running in a chosen direction, say, parallel to the rows.

To estimate the area of the COALESCER, we first assume to use full-tree COMBINERS, so that the side of the COMBINER has a length of  $O(t_i \log m_i)$  (parallel to the rows). Using Lemma 1 we have

$$\text{Height} = O(\sqrt{n_i} t_i \log m_i + n_i t_i) = O(n(1 + \frac{\log m_i}{\sqrt{n_i}})),$$

$$\text{Width} = O(\sqrt{n_i} t_i \log m_i) = O(n \frac{\log m_i}{\sqrt{n_i}})$$

The computation time is readily found as  $T_F = O(\tau + q + \log m_i)$ . We conclude

Lemma 3. An  $(n; t_{i-1}; t_i)$  full-tree COALESCER can be laid out in a rectangle  $O(n(1 + \log m_i / \sqrt{n_i})) \times O(n \log m_i / \sqrt{n_i})$  and operates in time  $T = O(\tau + q + \log m_i)$  ( $q$  is the input wordlength,  $n_i = n/t_i$ ,  $m_i = t_i/t_{i-1}$ ). When  $q = \theta(\log n)$ , then  $T = O(\log n)$ .

Similarly, Lemma 2 yields the following result:

Lemma 4. An  $(n; t_{i-1}; t_i)$  comb-tree COALESCER can be laid out in a rectangle  $O(n) \times O(n \log m_i / \sqrt{n_i})$  and operates in time  $T_C = O(\tau + q + m_i)$ . If both  $q$  and  $m_i$  are  $O(\log n)$ , then  $T_C = O(\log n)$ .

5.2. An Optimal VLSI Sorter. We now show that there is a COMBINATION-SORTER for words of length  $q = \theta(\log n)$  that sorts  $n$  elements in time  $T = O(\log n)$  and area  $A = O(n^2)$ , thus achieving the known lower bound for this problem. The sorter we propose is given by the block diagram in Figure 4. By the previous Lemmas 2 and 3 we see

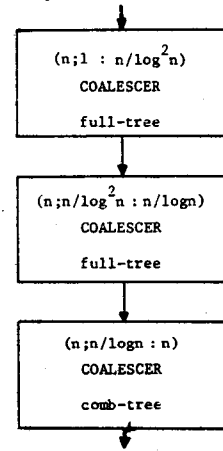


Figure 4. A COMBINATION-SORTER with three COALESCERS, for optimal VLSI sorting.

that the coalescers take area (width  $\times$  height)  $O(n) \times O(n)$ ,  $O(n \log \log n / \log n) \times O(n)$ , and  $O(n) \times O(n)$  respectively. It is also clear that the total time is  $O(\log n)$ . So we have:

Theorem 1. A VLSI sorter of  $n$  words of length  $q = (1+\epsilon) \log n$  can be constructed whose area and computation time are  $A = O(n^2)$  and  $T = O(\log n)$ . This network is  $AT^2$ -optimal in the chosen model of computation.

## 6. Area-Time Trade-Off

The COMBINATION-sorter proposed in the previous section has optimal area among sorters with minimum computation time. We will now describe a network, which, by choosing an appropriate value for a design parameter, allows us to sort in  $A = O(n^2 \log^2 n / T^2)$  for any time  $T \in [\Omega(\log n), O(\sqrt{n \log n})]$ . The network is the cascade interconnection of two components. The

first component is a COMBINATION-sorter for  $\frac{n}{s}$  inputs. The second component is a new network, called the mesh-of-CCC (MCCC) and obtained by suitably "hybridizing" known networks (the mesh and the CCC).<sup>(3)</sup> This network will now be described in detail.

An  $(n,s)$ -MCCC, with  $n = 2^v$ ,  $s = 2^\sigma$ , and  $r \triangleq n/s^2 = 2^\rho$  ( $\rho = v-2\sigma$ ) consists of  $s^2$  CCC modules, each with  $r$  cycles of length  $\rho$ . The  $n \times \rho$  processing elements of the MCCC are conveniently indexed as

$$M_{ij}(h,k): 0 \leq i, j < s, 0 \leq h < \rho, 0 \leq k < r.$$

For a fixed pair  $(i,j)$  the set  $\{M_{ij}(h,k): 0 \leq h < \rho, 0 \leq k < r\}$  is connected as a CCC-module, exactly as described in Section 3. Then CCC modules are arranged as an  $s \times s$  mesh, and, for a fixed  $k$ , the set of micromodules  $\{M_{ij}(0,k): 0 \leq i, j < s\}$  is mesh-connected (with  $i$  and  $j$  as row and column indices, respectively).

The MCCC graph can be laid out in a square of area  $O(n^2/s^2)$ , since each CCC requires  $O(n^2/s^4)$  area and channels of width  $O(n^2/s^2)$  allow a straightforward implementation of mesh-connections.

The mesh of CCCs has very interesting computing capabilities as illustrated by the following theorems, stated without proof:

Theorem 2. The MCCC can emulate the ASCEND (or DESCEND) paradigm [10] of the Binary-Cube in optimal  $AT^2 = \theta(n^2 \log^2 n)$  for any  $T \in [\Omega(\log n), O(\sqrt{n \log n})]$ .

Theorem 3. The MCCC can emulate the BITONIC SORTING paradigm [3] in optimal  $AT^2 = \theta(n^2 \log^2 n)$  for any  $T \in [\Omega(\log^3 n), O(\sqrt{n \log n})]$ .

To obtain networks faster than the MCCC we start from the following observation. A COMBINE-sorter with  $n/s$  input can sort (in time  $O(s \log n)$  and area  $O(n^2/s^2)$ )  $s$  sequences of  $n/s$  elements each. These sequences can then be fed, say one per column, into an MCCC with parameter  $s$ . At this point, the sequence in each CCC module is already sorted, and the MCCC is ready (after inverting the order of some sequences to comply with bitonic sorting rules) to execute the last  $2\sigma$  merging phases. (For the sake of simplicity we will ignore the fact that only  $\sigma$  phases would be really necessary after the work done by the COMBINE-sorter.) A simple analysis allows us to conclude that, in the process, the MCCC executes  $O(\log s + s)$  steps using  $O(\log n)$  time for each, thus running for a total time  $T = O(s \log n)$ . We can then state:

Theorem 4. A VLSI sorter of  $n$  words of length

<sup>(3)</sup>A similar network has been proposed by Aggarwal [16] who proves a variant of Theorem 2 given below.

$(1+\epsilon) \log n$  can be constructed with optimal  $AT^2 = \theta(n^2 \log^2 n)$  for any computation time  $T \in [\Omega(\log n), O(\sqrt{n \log n})]$ .

The reported research leaves the following open problems (among others):

1. Which are the upper and lower bounds to  $AT^2$  for sorting  $n$  words of length  $\leq \log n$  or considerably larger than  $\log n$ ?
2. Is there a suitable general framework of analysis encompassing number of words, length of key, and length of record?

#### Acknowledgement

We would like to thank Tom Leighton for invaluable discussions on parallel sorting.

This work was supported in part by the Joint Services Electronics Program under Contract N00014-79-C-0424 and in part by an IBM Predoctoral Fellowship.

#### References

1. C. D. Thompson, "The VLSI complexity of sorting," *IEEE Trans. Comp.*, vol. C-32, no. 12, Dec. 1983.
2. C. D. Thompson, A Complexity Theory for VLSI, Ph.D. Thesis, Computer Science Department, Carnegie-Mellon Univ., Aug. 1980.
3. G. Bilardi, F. P. Preparata, "A VLSI optimal architecture for bitonic sorting," Proc. 7th Conf. on Information Sciences and Systems, The Johns Hopkins University, Baltimore, MD, (March 1983); pp. 1-5.
4. K. E. Batcher, "Sorting networks and their applications," Proc. AFIPS Spring Joint Computer Conference, vol. 32, pp. 307-314, April 1968.
5. D. D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees," *IEEE Trans. Comp.*, vol. C-32, no. 6, pp. 569-581, June 1983.
6. F. T. Leighton, "New lower bound techniques for VLSI," Proc. 22nd Symp. on the Foundations of Computer Science, IEEE Computer Society, pp. 1-12, Oct. 1981.
7. D. E. Muller, F. P. Preparata, "Bounds to complexities of networks for sorting and for switching," *JACM*, vol. 22, pp. 195-201, April 1975.
8. F. P. Preparata, "New parallel sorting schemes," *IEEE Trans. Comput.*, vol. C-27, no. 7, pp. 669-673, July 1978.
9. M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, no. 5, pp. 458-473, May 1977.

10. F. P. Preparata, J. Vuillemin, "The cube-connected-cycles: A versatile network for parallel computation," Com. of the ACM, vol. 24, no. 5, pp. 300-309, May 1981.
11. D. Nassimi, S. Sahni, "Parallel permutation and sorting algorithms and a new generalized connection network," JACM, vol. 20, no. 3, pp. 642-667, July 1982.
12. F. T. Leighton, Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques, Ph.D. Thesis, Dept. of Mathematics, MIT, August 1981.
13. M. Aitai, J. Komlos, E. Szemerédi, "An  $O(n \log n)$  sorting network," Proc. 15th ACM Symposium on Theory of Computing, Boston, MA, April 1983, pp. 1-9.
14. F. T. Leighton, "Tight bounds on the complexity of parallel sorting," Proc. 16th ACM Symposium on Theory of Computing, Washington, D. C., April 1984.
15. G. Bilardi, F. P. Preparata, "The VLSI optimality of the AKS sorting network," Coordinated Science Laboratory, University of Illinois, Urbana, Report R-1008, UILU-ENG 842202, February 1984.
16. A. Aggarwal, "On I/O placement in VLSI circuits," Proc. 21st Annual Allerton Conference on Communication, Control, and Computing, Monticello, Illinois, pp. 236-243, October 1983.