Compact Grid Layouts of Multi-Level Networks *

S. Muthukrishnan † Mike Paterson ‡ Süleyman Cenk Şahinalp § Torsten Suel ¶

Abstract

We consider the problem of generating layouts of multi-level networks, in particular, switching, sorting, and interconnection networks, as compactly as possible on VLSI grids. Besides traditional interest in these problems motivated by interconnection topologies in parallel computing and switching circuits in telecommunications, there is renewed interest in such layouts in the context of ATM (Asynchronous Transfer Mode) switches. Our results improve on the existing area bounds for these networks by factors of up to three.

1 Introduction

We study the problem of laying out multi-level networks in general, and various switching, sorting and interconnection networks in particular, on a VLSI chip. The goal is to produce compact layouts, that is, layouts with the smallest possible grid area for realizing a given network. We present layouts for different types of "mappings" (formally defined later); they form building blocks for realizing any multi-level network. We use our building blocks and their special geometry, and combine them with high-level reorganization techniques to obtain (near-)optimal layouts for various sorting, switching and interconnection networks.

1.1 Specific Networks and Motivation

The specific networks that we consider are the butterfly, the Benes network, the bitonic sorting network, and general sorting networks. The basic elements in these networks are comparators or switches. These networks have been studied in many contexts (see [17, 19, 20, 26]). They are used as interconnection networks, or as switching networks in telecommunications. Motivated by the potential for massive parallelism, VLSI layouts of these networks have been studied extensively in the literature; see, e.g., [28, 20].

There is renewed interest now in layouts of switching and sorting networks on VLSI circuits within ATM (Asynchronous Transfer Mode), a promising network technology. Many ATM switches use the butterfly network and its related networks and sorters in order to route connections; see [30] for an overview, and [6] for generic ATM switch systems with such networks. Such ATM switches have been designed or developed at Bellcore (named *Sunshine* [4]), AT&T Bell Labs (named *Starlite* [13]), Lucent Technologies [16], Telecom Australia Research Labs [23], and elsewhere (for example, *Starburst* [31]). For high speed and performance, these ATM switches are fabricated as VLSI chips, with layouts based on Batcher's sorters, Benes networks, or banyan networks.

Much of the research on VLSI layouts of such networks has been inspired by the stringent need to be compact, and the focus has therefore been on trying to optimize the constant factor in the area of layouts, rather than obtaining merely big-O optimal bounds. Our results improve the area required by the best known constructions by factors ranging from 2 to 3. In practice, our results may translate to a smaller percentage of reductions in the actual chip area; however, even small reductions may be important.

1.2 The VLSI Grid Model

A layout of a network on an integer grid $\mathcal{G}_{R,C}$ is a mapping of the nodes of the network to grid points in $[1 \cdots R] \times [1 \cdots C]$. We concentrate on layouts whose input nodes are all on column 1 and whose output nodes are all on column C. The connections in the network are mapped to edge-disjoint paths on the grid. Two paths are allowed to cross at a grid point. A turn of a path at a grid point is called a *bend*.

Note that at most two paths can meet at a grid point, which implies that networks are restricted to containing devices of degree at most 4. All our communication networks are of this form, as they are based on 2-input, 2-output comparator and switching elements. When two paths meet at a grid point, the meeting may be *straight* or *knock-knee*: in the former, neither path changes direction at the intersection point, while in the latter, both paths change directions, that is, there are two bends. Our goal is to minimize the area, RC, of the layouts. The grid model for layouts was

*This research was partly supported by ESPRIT LTR Project no. 20244 - ALCOM–IT.

†AT&T Labs – Research, Florham Park, NJ, USA. Email: muthu@research.att.com.

‡Department of Computer Science, University of Warwick, Coventry, UK. Email: msp@dcs.warwick.ac.uk.

§Department of Computer Science, University of Warwick, Coventry, UK; and Center for BioInformatics, University of Pennsylvania, Philadelphia, PA, USA. Email: cenk@dcs.warwick.ac.uk.

¶Computer and Information Science, Polytechnic University, Brooklyn, NY, USA. Email: suel@photon.poly.edu.

To appear in the 31st ACM Symposium on Theory of Computing, Atlanta, GA, May 1999.

formalized in [27] for the VLSI setting.

1.3 Results for Specific Networks

We present the following optimal or near-optimal layouts for well-known networks, which improve upon the best previously known bounds.

Butterfly Networks. We present layouts of N-input butterfly switching/comparator networks of area[1] $(2/3)N^2$ and $(1/2)N^2$ for the cases when the order of the inputs and outputs is fixed or can be arbitrarily permuted, respectively; in both cases, the leading constant is the best possible.

The best previously known upper bound for the fixed-order case was the N^2 result that follows from the work of Wise [29]. (See also [1] for bounds when the inputs and outputs may be anywhere within the grid area.) Our first bound provides the standard interface of a stand-alone butterfly, while the second bound is useful when the butterfly is used as part of a larger circuit in which we can optimize the wiring. For example, using the second construction, we obtain a layout of an N-input dual Benes network with optimal N^2 layout area.

Batcher's Bitonic Sorting Network. We present a layout of Batcher's bitonic sorter that uses a total area of less than $1.39N^2$. Since these sorters are used in ATM switches, some attention has recently been given to producing compact layouts. However, the best previous layout of the bitonic sorter still required $3N^2$ area [8, 9], which improved upon earlier bounds in [7, 29].

Optimal Sorters. We address the question of determining the "best" sorter in general. There are known sorting networks of area N^2, but they use $\Omega(N^2)$ comparators [14], and thus the (logical) depth of the network is $\Omega(N)$. In contrast, Batcher's sorters use only $O(N \log^2 N)$ comparators with $O(\log^2 N)$ comparators on any path. A lower bound of N^2 for the area is known from [9].

We present a layout based on Columnsort that achieves N^2 area, and that can be used with any sorter as a subroutine. By combining Columnsort with a straightforward layout of the AKS sorter [2], we obtain a layout area of N^2 with $O(\log N)$ comparators on any path. Thus, this construction is optimal within a lower order additive term in the area, and asymptotically optimal in terms of the size and depth of the network. In fact, using the more relaxed definition of layout area in [7] the bound becomes $\frac{25}{32}N^2$, thus establishing a separation between the two different definitions of layout area.

1.4 General Techniques for Multi-Level Network Layouts

We develop the following general approach. First we design layouts of arbitrary networks of one or two levels. Here, a *level* is a permutation wiring connecting two "columns" of active elements. Then, by using these as local building blocks, we generate layouts of entire multi-level networks. Both these steps are nontrivial, and an overview follows.

Consider any network of one or two levels. We consider three different types of "mappings" from their inputs to their outputs (see Section 3). These mappings depend on whether the inputs and outputs are fixed or can be reordered arbitrarily, and on whether intermediate nodes may be arbitrarily placed within the layout or not. We use novel and existing algorithms to present layouts for each of these mappings which are tight up to at most two additional columns.

[1]Throughout this paper, we usually state only the leading term and suppress the additive $o(N^2)$ term.

Any multi-level network can be laid out by combining (concatenating) these mappings; however, this does not automatically lead to an optimal layout for the entire network. We use two additional techniques that lead to more compact layouts. The first technique is to rearrange a multi-level network in an isomorphic manner. For example, if the network compares a set of far-apart inputs in several successive levels, one may permute the input and then have several levels of comparisons between inputs that are close. We use such remappings of the inputs repeatedly in our layouts, and obtain significant savings in area. We point out that this idea is not really new, and that it has been used in the context of the butterfly to improve the locality of the FFT and bitonic sorting algorithms on parallel machines; see, e.g., [5, 24]. The second technique is to lay out building blocks (of one or two levels) into non-rectangular geometric shapes, which may then be fitted together more efficiently. We use this principle with triangular and parabolic blocks, and this too results in significant savings. The combination of these two ideas gives the best known results for the specific networks that we consider, and should also prove useful for producing tighter layouts of other multi-level networks.

We omit some details of the proofs and use figures and geometric intuition to illustrate some of our ideas. At the beginning of his paper [29], Wise says, "This paper offers two results that can both be described as pictures. They are Figures 1 and 3. The perceptive readers may stop here, since the remainder of this paper only describes them." The same is not quite true here since our structured approach to deriving our layouts should be of independent interest. For example, the problem of laying out one of the "mappings" for a level turns out to be the problem of finding disjoint paths on grid graphs – another problem with a lot of history; see, e.g., [10, 12, 11, 15, 32].

Note that all our constructions use knock-knees. Knock-knees have been used in many other papers on VLSI layouts (see Chapter 9 in [20], and [21]). From the VLSI technology point of view, the horizontal and vertical lines of the grid model are laid out on two different layers. Bends, knock-knees and switches all operate on both layers and hence, bends and knock-knees are no harder to fabricate than switches. Nevertheless, like switches, bends and knock-knees are also a technological resource, and the number of bends (and hence the number of knock-knees) in the constructions that we use as building blocks for the layouts of switching and sorting networks is a small constant per connection.

2 Preliminaries

Given $M = 2^m$ for some $m \geq 1$, a *butterfly interconnection network* $\mathcal{B}^m$ has M input nodes $I^m[0, \ldots, M-1]$, and M output nodes $O^m[0, \ldots, M-1]$, and consists of m levels, $\mathcal{B}^m_m, \mathcal{B}^m_{m-1}, \ldots, \mathcal{B}^m_1$. Each level $\mathcal{B}^m_\ell$ is a bipartite graph with M input and M output nodes. Each input node $I^m_\ell[i]$ of $\mathcal{B}^m_\ell$ is connected to two output nodes $O^m_\ell[i]$, and $O^m_\ell[\bar{i}^\ell]$, where $\bar{i}^\ell$ is obtained by complementing the ℓth least significant bit of i.

The butterfly interconnection network can be used as a building block in switching (or comparator) networks if each node is replaced by a two-input, two-output switch (or comparator). In particular, this implies that the inputs and outputs now each consist of $N = 2M$ lines. Such a network is called a *butterfly switching* (*comparator*) network. Switching networks of particular interest are the Benes network and its dual. A *Benes network* of $N = 2M$ input and output lines is a concatenation of butterfly levels $\mathcal{B}_m^m, \mathcal{B}_{m-1}^m, \ldots, \mathcal{B}_1^m, \mathcal{B}_1^m, \mathcal{B}_2^m, \ldots, \mathcal{B}_m^m$ in that order, that is, it consists of two complete butterfly networks connected end to end, in which the second butterfly has reversed levels. The *dual Benes* is formed similarly by a reversed butterfly followed by a butterfly. These Benes networks are of interest since they are rearrangeable (see, for example, [26]). We omit the standard definition of Batcher's bitonic sorter; see, e.g., [3, 17].

3 Optimal Basic Layouts

In this section, we identify three layout problems defined by "mappings". These mappings model one or two levels of wiring and comparators (or switches) in a multi-level network. The mappings model: (1) an arbitrary wiring from the outputs of one level of comparators to the inputs of the next, when the comparators at each level are fixed within their respective grid columns (Section 3.1); (2), the same as (1), but the comparators at the second level may be moved arbitrarily within their grid column (Section 3.2); and (3) an arbitrary wiring from the outputs of level ℓ to the inputs of level $\ell + 2$, when the comparators at levels ℓ and $\ell + 2$ are fixed within their respective grid columns, but the comparators at level $\ell + 1$ may be moved around arbitrarily (Section 3.3).[2]

3.1 Permutation Networks

This problem is defined as follows. On an integer grid, we have n inputs and n outputs, whose rows are specified as $r_1, \ldots, r_n$ and $s_1, \ldots, s_n$ respectively. Let f denote such an input-output specification and a permutation on $\{1, \ldots, n\}$. The goal is to produce a layout of this permutation (that is, connect each input i to output $f(i)$ via a set of edge-disjoint paths) such that all inputs, and all outputs, respectively, are on the same column. Such a network is called a *permutation network*.

Given f, define the *cutwidth under row* i, denoted $C_i(f)$, as $|\{j \mid r_j \le i < r_{f(j)} \text{ or } r_{f(j)} \le i < r_j\}|$. The *cutwidth of the permutation* f, denoted by $C(f)$, is defined as $\max_i C_i(f)$. We have $C(f) \le n$. The following holds (see [10] for a proof): any grid layout of f needs at least $C(f)$ columns, independent of the number of rows used. Since any nontrivial permutation on n inputs requires at least $n + 1$ rows, we get a lower bound on the area of any layout of at least $(n + 1)C(f)$.

The best known algorithms to lay out a given permutation network in optimal area run in $O(n \cdot C(f))$ time and result in $O(n \cdot C(f))$ bends [10, 32]. What we show below is that for any permutation one can construct a layout which results in only $O(1)$ bends per connection; for some of the permutations used in our switching and sorting networks, our layout results in optimal $(n + 1)C(f)$ area. To simplify the rest of the discussion we assume that all inputs and outputs are located on n contiguous rows of the grid, although our results apply to all possible row placements of inputs and outputs.

Faster construction, using fewer bends. We consider layouts in which we limit the number of bends used. For all but the simplest permutations, most of the connections require at least two bends, independent of the number of rows and columns used. In what follows, we show layouts in which we use at most four bends per connection.

Given a permutation f on $\{1, \ldots, n\}$, consider the directed graph $G(f)$ with vertices $V = \{1, \ldots, n + 1\}$ and edges $E = \{(i, f(i) + 1) \mid 1 \le i \le n\}$. In $G(f)$, every vertex except $n + 1$ has outdegree one and every vertex except 1 has indegree one. For $n + 1$ the outdegree is zero, and for 1 the indegree is zero. Clearly, if $G(f)$ has a Hamiltonian path, then that path goes from 1 to $n + 1$.

In $G(f)$, a path p is *maximal decreasing (maximal increasing)* if the sequence of vertex numbers along p is decreasing (increasing) and no extension of p has this property.

We are interested in the total number of maximal decreasing and maximal increasing paths, which we denote by $m(f)$; trivially, $m(f) \le n$. The following result proves useful since $m(f) \approx C(f)$ for a number of permutations that we consider later.

Theorem 1 *There is an $O(n)$ time algorithm to produce a layout of f using $n+1$ rows and at most $m(f)+2$ columns. Each connection has at most four bends.*

Proof: The layout uses rows $1, \ldots, n + 1$, where row i contains input node I_i and output node O_i. We use the graph $G(f)$ to determine an assignment of columns to connections so that no two paths assigned to connections overlap. The layout is in two parts: the first part produces its outputs $O'_1, \ldots, O'_n$ in rows $2, \ldots, n + 1$ respectively, while the second part uses a single column to restore the outputs to their proper final positions.

If $G(f)$ has (i.e., is) a Hamiltonian path, the column assignment to connections is as follows. Consider the maximal increasing path ending at $n+1$, and starting at some j_1. For each edge $(i, f(i) + 1)$ on this path, we assign column 1 to the connection from I_i to $O'_{f(i)}$ (in row $f(i) + 1$). Next we find the maximal decreasing path ending at j_1, and starting at some j_2. We assign column 2 to the connections corresponding to edges on this path, and so on. The structure of $G(f)$ ensures that this procedure proceeds smoothly and the resulting assignment is non-conflicting; the number of columns used is clearly $m(f)$. The output connections can now be shifted back one row using a single extra column.

If $G(f)$ does not have a Hamiltonian path, we determine an intermediate permutation g, such that (1) $G(g)$ has a Hamiltonian path, (2) $m(g) \le m(f) + 1$, and (3) f is the composition of g and a permutation h, which has a layout that uses a single column and which restores the outputs to rows $1, \ldots, n$.

The permutation g is computed as follows. Consider the path p in $G(f)$ that starts from vertex 1 and ends at vertex $n + 1$. If $G(f)$ is not Hamiltonian there are nodes which are not on this path. Because each such node has both indegree and outdegree 1, it has to be on a cycle. Our method relies on incorporating each such cycle into path p as follows. We find the maximum j which is not on path p and incorporate it into the path as follows. The edges $(k, j + 1)$ and (k', j) in E are replaced by the new edges $(k', j + 1)$ and (k, j). Thus the path from 1 which went along edge $(k, j + 1)$, now takes the detour k, j, then around the cycle that contained (k', j),then $k', j + 1$. This corresponds to swapping the neighboring destination outputs $j - 1$ and j, i.e., whereas $f(k) = j$ and $f(k') = j - 1$, we have a new permutation f' where $f'(k) = j - 1$ and $f'(k') = j$.

After this swap, we claim that all nodes on the original cycle of node j will now be on p. We iteratively apply swaps until all cycles are incorporated into p. This new graph G' provides the permutation g and is Hamiltonian. Because

[2] An alternative problem considered in [12] allows flexibility on the rows the input and outputs are located, provided that their relative orderings do not change. Computing an optimal area layout for this problem is NP-hard [12].

the swaps are local, we can show that $m(g) \le m(f) + 1$, and that the permutation h can be laid out using a single column, since we only need to reorder the swapped output nodes, and these do not overlap. ∎

There are permutations f that have layouts using fewer than $m(f)$ columns; hence, our construction is not always optimal in area. Nevertheless, we will see that the construction above is optimal for certain important classes of permutations that are of particular interest to us (see Section 4).

3.2 Pairing Network

We are given a grid with the inputs on the left side numbered $I_1, I_2, \ldots, I_{2M}$, and the outputs on the right side numbered $O_1, O_2, \ldots, O_{2M}$, where I_i and O_i are in row i. A set of M disjoint input pairs (lu_i, ld_i), where $lu_i, ld_i \in \{I_1, \ldots, I_{2M}\}$ is given. The goal is to determine a one-to-one mapping f of $I_1, \ldots, I_{2M}$ to $O_1, \ldots, O_{2M}$ such that, for each i, $f(lu_i)$ and $f(ld_i)$ are adjacent (that is, they differ by one); furthermore, we must find a layout of the permutation network f. Such a network is called a *pairing network*. We write $N = 2M$.

Consider the graph G on a linear array of N vertices, in which there is an edge between nodes i and j if and only if (I_i, I_j) is an input pair for the pairing network; thus there are M edges in all. The *pair-cutwidth* $P(G)$ of the input instance is the maximum number of crossing edges of G between any pair of rows i and $i+1$.

Theorem 2 *In any layout of a pairing network, the number of columns used is at least $P(G) - 1$, independent of the number of rows used. There is an algorithm to produce a layout of any pairing network in which the number of rows is at most $N+1$ and the number of columns is at most $P(G)+3$; this algorithm takes time $O(N \cdot P(G))$.*

Proof: For the lower bound, we consider any i such that $P(G)$ edges of G cross between rows i and $i+1$. Each edge corresponds to a pair of inputs, of which at least one needs to be taken across a vertical edge between rows i and $i+1$ to bring them together, with the possible exception of one pair which could be mapped to rows i and $i+1$.

For the upper bound, we first produce a mapping f that satisfies the requirement that the cutwidth of f is nearly the same as the pair-cutwidth of the input. Then, we apply one of the algorithms for laying out a permutation in optimal area. For the first step, Lemma 3 establishes the bounds needed. ∎

Lemma 3 *For any input to the pairing network, there is an $O(N)$ algorithm to find a satisfying permutation f such that $C(f) \le P(G) + 1$.*

Proof: We construct f iteratively. Here, we simply outline the algorithm. Consider the pair (lu_i, ld_i) where (without loss of generality) $lu_i = 1$. We assign $f(1) = 1$ and $f(ld_i) = 2$. Now we have two cases. If ld_i is even, we consider the pair (lu_j, ld_j) where (without loss of generality) $lu_j = ld_i - 1$; we assign $f(lu_j) = ld_i - 1$ and $f(ld_j) = ld_i$ and continue iteratively with ld_j. On the other hand, if ld_i is odd, we consider the pair (lu_j, ld_j) where (without loss of generality) $lu_j = ld_i + 1$; we assign $f(lu_j) = ld_i$ and $f(ld_j) = ld_i + 1$ and continue iteratively with ld_j. We can set up an appropriate directed graph, with in- and outdegrees being one, on which this process corresponds to a walk; since such a graph has a cycle decomposition, this iterative process goes smoothly and terminates at row 2. If the graph is not Hamiltonian, we find the topmost input row which is not assigned an output row and iterate. The argument that $C(f) \le P(G) + 1$ can be outlined as follows — f essentially realizes a pairing network in which at most one endpoint of each input pair is changed by one: the output rows of a pair (lu_j, ld_j) are within $(lu_j - 1, ld_j + 1)$. The pair-cutwidth of the graph G' that is obtained from this pairing is at most $P(G) + 1$. ∎

3.3 Pair-Permute Network

For the grid as before, with I_i and O_i in row i, a set of M disjoint input pairs (lu_i, ld_i), for $lu_i, ld_i \in \{I_1, \ldots, I_{2M}\}$ is given, with the M associated disjoint output pairs (ru_i, rd_i), where $ru_i, rd_i \in \{O_1, \ldots, O_{2M}\}$. Our goal is to determine the layout of a network in which each input pair is connected to a comparator (or switch), the outputs of which are connected to the output pair. The entire network is called the *pair-permute network*. We set $N = 2M$.

Consider the graph G on a linear array of N vertices in which, for each input-output 4-tuple $(lu_i, ld_i, ru_i, rd_i) = (I_i, I_j, O_k, O_l)$, we put edges between vertices $\max\{i, j\}$ and $\min\{k, l\}$ and between vertices $\min\{i, j\}$ and $\max\{k, l\}$; thus there are N edges in all. The *pair-permute-cutwidth* of the input instance, $PP(G)$, is the maximum number of crossing edges of G between any pair of rows i and $i+1$.

Theorem 4 *Any layout of a pair-permute network requires at least $PP(G)$ columns, independent of the number of rows used. There is an algorithm to compute a layout of any pair-permute network using at most $PP(G)$ columns and $N+1$ rows; this algorithm takes time $O(N \cdot PP(G))$.*

We sketch the proof here. Consider any input-output 4-tuple (I_i, I_j, O_k, O_l). We define a mapping f as follows: $f(I_{\max\{i,j\}}) = O_{\min\{k,l\}}$ and $f(I_{\min\{i,j\}}) = O_{\max\{k,l\}}$. For this f, we apply any of the algorithms for laying out a permutation in optimal area. This produces a layout in which the two routes from the input pair to the output pair cross, and we can put the intermediate comparator (or switch) at that intersection. This completes the construction. We claim that $C(f)$, the cutwidth of f, is at most $PP(G)$; this requires a proof that is not very difficult, and it is omitted here.

4 Layouts of Classes of Permutations

In the constructions in Section 5, we use two special classes of permutation, called *bit reversal* and *unshuffle* permutations. In this section, we define these classes and describe their properties.

Definition 4.1 *Let $N = 2^n$ and $k \le n$. The* k-bit reversal permutation *(or* k-bit reverser*) is the permutation on $\{0, \ldots, N-1\}$ that connects each element x with the element x' obtained by reversing the order of the k least significant bits of the binary representation of x. E.g., the 3-bit reversal of* 1011 *is* 1110.

Definition 4.2 *Let $N = 2^n$ and $k \le N$, and assume for simplicity that N/k is an integer.*

(a) The k-way unshuffle permutation *is the permutation that connects each x with $x' = (x \bmod k) \cdot \frac{N}{k} + \left\lfloor \frac{x}{k} \right\rfloor$.*

(b) The k-way shuffle permutation *is the permutation that connects each x with $x' = (x \bmod \frac{N}{k}) \cdot k + \left\lfloor \frac{kx}{N} \right\rfloor$.*

Note that a k-way shuffle permutation is equivalent to an $\frac{N}{k}$-way unshuffle permutation. If $k = 2^r$, then a k-way shuffle (unshuffle) permutation corresponds to a rotation of the bit representation of each position by r positions to the right (left).

Definition 4.3 *A* bit-permutation *is a permutation on a set of elements induced by a permutation on the bits of their binary addresses.*

For example, a bit-reversal or any 2^r-way shuffle or unshuffle is a bit permutation.

We now consider the area and shape of layouts of some of these permutations. Given a partitioning of N elements into N/b disjoint blocks with b elements each, we say that a permutation is an *all-to-all permutation* with block size b, $b \geq \sqrt{N}$, if exactly b^2/N elements in block i are connected to elements in block j, for all i, j. It is easily seen that the n-bit reverser on $N = 2^n$ elements, as well as any k-way shuffle and unshuffle with $k = \omega(1)$ and $k = o(N)$, is an all-to-all permutation for some block size $o(N)$. Also, for a given block size, any all-to-all permutation can be reduced to any other all-to-all permutation by performing appropriate local permutations inside the blocks, resulting in at most a lower order additive difference in layout area.

It is easily shown that there exists an all-to-all permutation π for which $G(\pi)$ defined in Section 3.1 has a Hamiltonian path, and hence Theorem 1 can be applied. By performing an additional analysis of the shape of the resulting layout, we can show the following lemma.

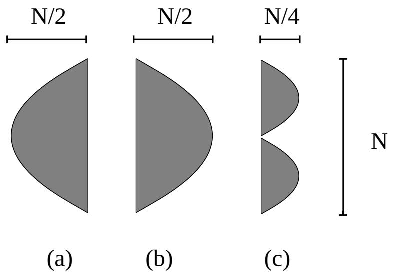

Figure 1: Layout shapes of (a) a left-oriented and (b) right-oriented k-way unshuffle or n-bit reverser, and (c) of a (right-oriented) $(n-1)$-bit reverser.

Lemma 5 *The n-bit reverser and any k-way unshuffle or shuffle with $k = \omega(1)$ and $k = o(N)$ can be laid out in $N/2 + o(N)$ columns, with only a constant number of bends on any path. The layout has a parabolic shape such that row i of the layout occupies only $2i(N-i)/N + o(N)$ columns.*

We can use both left-oriented and right-oriented parabolas, as shown in parts (a) and (b) of Figure 1, respectively. Note that a k-bit reverser with $k < n$ is equivalent to applying a k-bit reverser separately to each of the 2^{n-k} disjoint blocks of 2^k elements. Thus, for $k = n - 1$ one possible layout has depth $N/4$ and a shape as shown in Figure 1(c).

5 Specific Networks

5.1 Butterflies

We present a triangular layout for the butterfly switching network with $N = 2M = 2^{m+1}$ inputs.

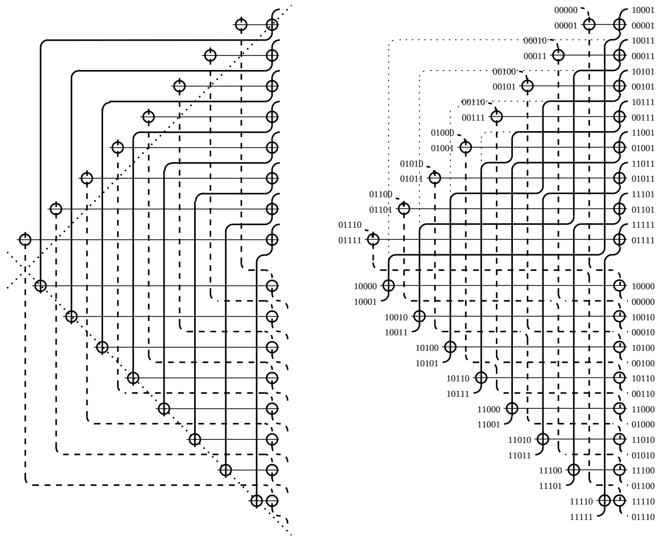

Figure 2: Preliminary outline and triangular layout of a $\mathcal{B}_4^4$.

The top level $\mathcal{B}_m^m$ of a butterfly network $\mathcal{B}^m$ transfers half of the 2^m inputs in the top half to the bottom half, and vice versa, thus requiring width at least M at the midpoint. A more detailed analysis shows that, for $1 \leq j \leq M$, the width required is at least j for the rows at distance j from the top or the bottom. The construction illustrated in Figure 2 achieves this optimal width together with the optimal height, both to within an additive constant. On the left in Figure 2, we show a simple preliminary layout. The final layout is obtained by folding in the lefthand corners along the dotted lines. For the smaller levels, $\mathcal{B}_\ell^m$ where $\ell < m - O(1)$, we need to use rectangular layouts which can be packed tightly in a vertical stack without requiring too many extra rows.

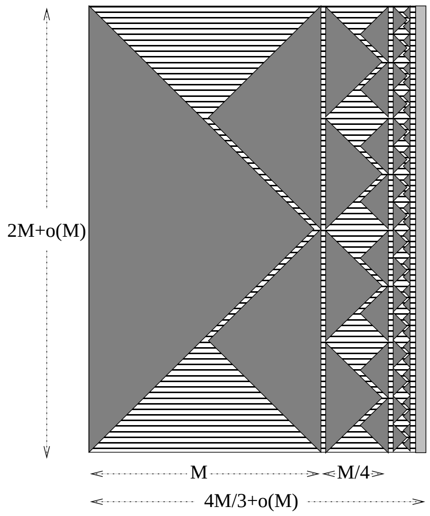

Figure 3: Sketch of layout of a butterfly network using triangles.

The triangular layout yields a considerable advantage in laying out a complete butterfly network. We can lay out a $\mathcal{B}_{m-1}^m$ in the same way, as a pair of half-size triangles one above the other. If the layout is done with the triangles pointing the opposite way from that of the $\mathcal{B}_m^m$ then the two layouts can be fitted together snugly within about M columns. Similarly, the $\mathcal{B}^m_{m-2}$ and the $\mathcal{B}^m_{m-3}$ fit together in about $M/4$ columns. The total width required for the complete butterfly network is therefore only $M(1 + 1/4 + \cdots) \approx 4M/3 = 2N/3$. See Figure 3.

Theorem 6 *The optimal area for a layout of the butterfly switching network with N inputs is $2N^2/3 + o(N^2)$.*

Proof: A construction for the upper bound was described above. For a matching lower bound on the layout area, we make use of one property of the butterfly comparator network which is used when it occurs as a component of a bitonic sorter. It can perform a merge of a pair of sorted sequences, when they are presented at the inputs in interleaved fashion, with their sorted orders running in opposite directions. As a consequence, for any i, the input sequence $1^{(N-2j)}(10)^j$ (an interleaving of $1^{N/2}$ and $1^{N/2-j}0^j$) can be transformed into the output sequence $0^j1^{(N-j)}$.

Consider any layout of a butterfly comparator (or switching) network. We can find a row r such that i inputs and j outputs appear above r, and $N-1 \leq i+2j \leq N+1$. From the above merging property, we can see that at least $i+j$ edge-disjoint paths cross from row $r-1$ to row r, i from above carrying 1's and j from below carrying 0's. If $i > j$ then $i + j > 2i/3 + 4i/3 \geq 2(N-1)/3$, giving a lower bound of $\lfloor 2N/3 \rfloor$ columns. Similarly we could look for a row r' with i' inputs and j' outputs occurring *below* r' and $i' + 2j' \approx N$. If $i' \geq j'$ then the $\lfloor 2N/3 \rfloor$ lower bound follows as before.

Let us suppose therefore (without loss of generality) that $j - i \leq j' - i'$, and that $j - i = a \geq 0$. It is easy to see that there are at least $2a$ rows between r and r' not containing outputs, and so the height of the layout is at least $N + 2a$. The area A satisfies

$$A \geq (N+2a)(i+j) \geq (N+2a)(2N-2-a)/3 \geq 2N^2/3+o(N^2)$$

provided that $a \leq 3N/2$. Values of a above $3N/2$ cannot give any lower area since we have an independent lower bound on the width of $N/2$ from Theorem 7. ∎

5.2 Permuted Butterflies and Dual Benes Networks

Sometimes we can tolerate a permutation of the input nodes or the output nodes of a layout to save space. In a standard layout of a butterfly interconnection network, the total area is dominated by the first few levels $\mathcal{B}^m_m, \mathcal{B}^m_{m-1}, \ldots$, and the contribution of the second half of the sequence of m levels is negligible. Consider the effect of permuting the order of the M input nodes using the m-bit reversal permutation. Now, the initial levels can be laid out as $\mathcal{B}^m_1, \mathcal{B}^m_2, \ldots$, and the contribution of the first half is negligible. At the middle of the layout we restore the original order of the nodes, so that now the second half of the layout retains its negligible area. The required permutation of the $2M$ lines is the reversal of the leading m bits, which can be laid out like the m-bit reverser but with pairs of adjacent lines being routed in parallel. We find then that the total area of this layout is dominated by the bit permutation in the middle. Only $M + o(M)$ columns are required, with a rectangular area of about $2M^2 = N^2/2$. We will later take advantage of the parabolic shape of the layout.

The dual Benes network consists of a left-to-right-reversed butterfly network followed by a normal butterfly. So the corresponding juxtaposition of two input-permuted butterflies also gives the same network, since the two permutations meeting in the center cancel each other. Hence the dual Benes network can be laid out in an area of about N^2.

Theorem 7 *(i) The optimal area for a layout of the dual Benes network with N inputs is $N^2 + o(N^2)$ and at least $N-1$ columns are required in any layout.*
(ii) The optimal area for a layout of a permuted butterfly switching network with N inputs is $N^2/2 + o(N^2)$ and at least $N/2$ columns are required in any layout.

Proof: The constructions are described above. The lower bounds for the Benes network follow from the rearrangeability property of this network. For any layout we can find a row r such that i inputs and j outputs lie above r, and $N-1 \leq i+j \leq N$. Since the Benes network can permute the input sequence 1^i0^{N-i} to the output sequence $0^{N-i}1^i$, the lower bound of $N-1$ columns is immediate.

The lower bounds for the permuted butterfly switching network follow from our construction of the Benes network using a pair of adjacent butterfly networks. ∎

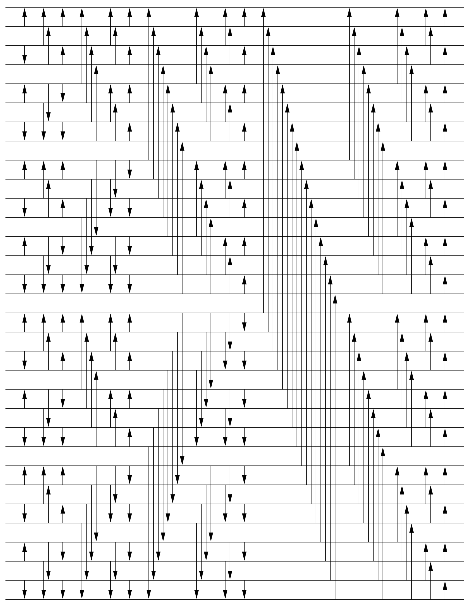

Figure 4: A bitonic sorting network for 32 inputs.

5.3 Bitonic Sorting Networks

A conventional comparator network for bitonic sort with $N = 2^n$ inputs can be represented, as for example in [17], with 2^n horizontal lines representing the inputs, and comparators shown as arrows linking the pairs of inputs which are to be compared at each time step (see Figure 4). The network consists of n successive merging phases. In the ith phase, for $1 \leq i \leq n$, pairs of sorted sequences of length 2^{i-1} are merged. In each pair the two sequences are presented in oppositely sorted order.

A naive layout of this network would involve bringing into adjacent lines the pairs of inputs to be compared and then returning them in the appropriate order to their original pair of positions. One obvious improvement would be to bring the required inputs together but not to return them afterwards, merely remembering their logical positions. We adopt an alternative strategy here. Any comparator layer corresponding to lower-order bit positions requires only few columns. We introduce bit permuters at suitable places in the layout so that *every* comparator layer now corresponds to a low-order bit. The layout area is now dominated by the layout of the bit permuters.

As illustrated in Figure 4, the sequence of bit positions (the bits running from 1 (low) to n (high)) corresponding to the sequence of comparator layers is:

$$1; 2, 1; 3, 2, 1; \ldots; n-1, \ldots, 3, 2, 1; n, \ldots, 2, 1.$$

The ";"s mark successive merging stages. Corresponding to bit $i+1$, we use a $\mathcal{B}_i^{(n-1)}$ to bring together inputs with addresses that differ in the $(i+1)$st bit. In terms of bit permutations, the $(i+1)$st bit and the first bit are exchanged. Then a line of comparators is used, followed by another $\mathcal{B}_i^{(n-1)}$ to restore the previous order.

We choose k such that $\sqrt{n} > k \geq 2\log n + \omega(1)$, so that $n^2 2^{n-k} = o(N)$ and $k^2 < n$. By using bit permuters we will ensure that we only require $\mathcal{B}_\ell^{(n-1)}$'s for $\ell \leq n-k$ and the choice of k ensures that the total width of all these butterfly levels is $o(N)$. We will need two n-bit reversers, one $(n-\ell)$-bit reverser for each of $\ell = 1, 2, 3, \ldots k$, and in addition $O(n)$ smaller bit reversers. The results of Section 4 show that the total width of all of these is only $N(1/2+1/2+1/4+1/8+\cdots) + o(N) = 3N/2 + o(N)$.

In the following outline of the construction we will refer to component parts with $O(N/2^k)$ columns as *small* and other components as *large*. Described *from right to left*, our layout first uses $n-k$ small butterfly levels, but then a bit permutation is used to interchange the k bits $(n-k+1, \ldots, n)$ about to be encountered with the k bits most recently dealt with, i.e., $n-2k+1, \ldots, n-k$. This bit permutation is easily achieved using a constant number of small bit reversers and one n-bit reverser. Now, the only large butterfly levels are those operating on (what were) bits $n-2k+1, \ldots, n-k$, and so the next $n-k$ levels are small. When a large level is about to be encountered again, another bit permutation is used to move the most recently processed bits into the top k positions. This involves a second n-bit reverser.

This seems to be becoming expensive, but now a small adjustment allows us to meet the claimed bound. In the second bit permutation from the right, we move bit n back into the most significant position, as well as moving the $k-1$ most recently encountered bits into the other $k-1$ most significant positions. The point of this is that bit n is used in a butterfly level just once, at the beginning of the final merge phase. Therefore at this point, running backwards through the network, there is no further use of this bit. It can remain always now in the most significant position, and the width of an $(n-1)$-bit reverser is only about $N/4$. In a similar way at the third-last bit permutation, bit $n-1$ can be lodged permanently in the second-most-significant position, reducing the width of the next large bit reverser to only $N/8$, and so on.

One further detail needs attention. The rightmost bit permutation comes just to the left of bit $n-k$ in the final merge phase, the next bit permutation comes to the left of bit $n-2k$ in the previous merge phase, the next is to the left of bit $n-3k+1$, and so on. Since $k^2 < n$, there is space for at least k phases in this pattern before a change is needed. Beyond this point (to the left of the final k merge phases) all butterfly levels will be small, since the k most significant bits are in their natural positions and are not used by the comparators.

A Further Improvement. The $1.5N^2$ bound of the previous subsection was obtained by simply adding up the number of columns needed for each of the bit reversers. We now show how the layout area can be further improved to about $1.39N^2$ by "packing" the components in a more space-efficient way. To do this, we need to exploit the parabolic shape of the bit reverser layouts shown in Figure 1.

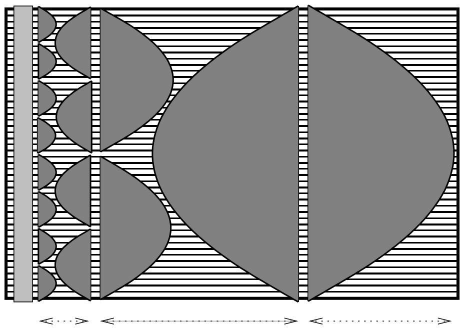

Figure 5: A layout of the bitonic sorting network with area $\frac{25}{18}N^2$.

To get the improved bound, we lay out the last of the two n-bit reversers in the straightforward way, We then group the other n-bit reverser with the $(n-1)$-bit reverser, and the $(n-2)$-bit reverser with the $(n-3)$-bit reverser, and so on, and lay out each of the groups in a more area-efficient way by using the left-oriented and right-oriented layouts of the bit reverser, as introduced in Section 4.

Recall that an n-bit reverser can be laid out such that the ith row of the layout uses $2i(N-i)/N$ columns. By simple calculus, we can show that we can lay out a right-oriented $(n-1)$-bit reverser followed by a left-oriented n-bit reverser in a total of $\frac{2}{3}N$ columns (instead of the trivial $\frac{3}{4}N$). For the group containing the $(n-3)$-bit reverser and the $(n-2)$-bit reverser, we get a layout with $\frac{2}{12}N$ columns, and continuing this we get a total of

$$\frac{1}{2}N + \sum_{i=0}^{n} \frac{2}{3 \cdot 4^i} N = \frac{25}{18} N$$

columns for the entire network. The resulting overall structure is shown in Figure 5, and we get the following result.

Theorem 8 *The bitonic sorting network can be laid out in area* $\frac{25}{18}N^2 + o(N^2)$.

5.4 A Sorting Network with Optimal Layout Area

In this section, we describe a layout of a sorting network based on the Columnsort algorithm [18] with area $N^2 + o(N^2)$. We first sketch the Columnsort algorithm, and then describe the layout of the corresponding sorting network.

5.4.1 Columnsort in a Nutshell

Columnsort is a simple parallel sorting algorithm proposed by Leighton [18]. The basic idea is to reduce the problem of sorting N elements to that of (repeatedly) sorting subsets of $N^{2/3}$ elements.

In the following, we assume that the input is given as an array $A[0 \ldots N-1]$ of size N, on which the algorithm operates by means of comparisons and permutations. For simplicity we assume that $N = B^3$ for some integer B. We think of A as being partitioned into B blocks of size B^2, where block i consists of $A[i \cdot B^2]$ to $A[(i+1) \cdot B^2 - 1]$, and we refer to $A[i \cdot B^2 + j]$ as element j of block i. Then the algorithm runs in the following six steps:

(1) In each block, sort the B^2 elements into ascending order.

(2) Perform a B-way unshuffle permutation on A, moving element $Bj + k$ of block i to element $iB + j$ of block k, for all $i, j, k \in [0 \ldots B-1]$.

(3) In each block, again sort the B^2 elements into ascending order.

(4) Perform a B-way shuffle permutation on A, moving element $iB + j$ of block k back to element $Bj + k$ of block i.

(5) In each block, again sort the B^2 elements into ascending order.

(6) Perform two merging steps, first between blocks $2i$ and $2i+1$, for $0 \leq i < B/2$, and then between blocks $2i-1$ and $2i$, for $0 < i \leq B/2$.

For a proof of correctness, we refer the reader to [18]. For some basic intuition, observe that Step (2) distributes the elements of each sorted block in a round-robin fashion among all B blocks. As a result, each block receives an "approximately evenly" spaced subset of all input elements. This means that after Step (3), the relative position of each element inside a block can be used to estimate an approximate destination block, to which it is routed in Step (4). At this point, it can be shown that every element is at most one block away from its final destination, and hence Steps (5) and (6) suffice to finish the sort.

5.4.2 An Efficient Layout

Our layout of Columnsort implements the six steps of the algorithm from left to right. Each element of the array A corresponds to a row of the layout.

Note that a variety of algorithms can be used to implement the sorting in Steps (1), (3), and (5), for example AKS, bitonic sort, or the present algorithm used recursively. In all of these cases, the number of columns needed for the layout of these steps is at most $O(N^{2/3}\text{polylog}(N))$ even under very naive layouts. The same is also true for the merging networks needed for Step (6).

Thus, the layout area of the network is determined by the layout area of the permutations that are routed in Steps (2) and (4). Since these unshuffle and shuffle permutations can be implemented in $N/2 + o(N)$ columns, we get the following result if we use AKS in Steps (1), (3), and (5).

Theorem 9 *There exists an $O(\log N)$ depth sorting network with layout area $N^2 + o(N^2)$.*

We point out that this matches the N^2 lower bound for the area of any sorting network shown by Even [7] to within an additive lower order term. We note that the area inside the bounding rectangle that is actually occupied by our layout is even smaller than that, $\frac{2}{3}N^2 + o(N^2)$, and that it has the shape of a symmetric parabolic lens, formed predominantly by the juxtaposition of two opposite parabolic layouts of bit reversers.

It can also be shown that this Columnsort layout achieves an area of $\frac{25}{32}N^2 + o(N^2)$ under the more relaxed definition of layout area considered in [7], where the bounding rectangle containing the circuit does not have to be aligned with the coordinate axes. For this definition of layout area, Even [7] has proved a lower bound of $\frac{1}{2}N^2$, and hence our result establishes a separation between these two definitions of area complexity. See Figure 6 for an illustration of this layout.

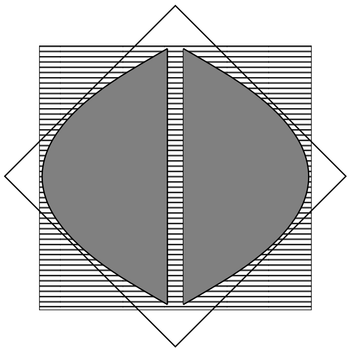

Figure 6: Layout of Columnsort using two all-to-all permutations. Shown is the standard layout area of N^2 as well as the layout area of $\frac{25}{32}N^2$ if the bounding box does not have to be parallel to the axes.

6 Extensions and Discussions

Due to space constraints, we have to leave a detailed discussion of our results to the full paper. Here, we only mention briefly several extensions and open problems.

- Our results on butterflies can be used to obtain tight bounds for the layout of merging networks. If the two sorted lists are input into the circuit in an interleaved fashion, then about $N/2$ columns suffice; this is also the best possible under any ordering of inputs and outputs. If the two lists are supplied separately, then about N columns are needed.

- An interesting open problem is whether we can get tight bounds for the bitonic sorter, and for general sorting under the relaxed definition of VLSI layout area assumed in [7]. In the latter case, the problem boils down to closing the gap between our back-to-back parabolic upper bound and the diamond-shaped lower bound given by the argument in [7].

- If the comparators are larger than unit size, say, occupying a $(k \times k)$ area, then we can replace each "layer" of $O(N)$ comparators by $O(N/r)$ successive layers with r comparators in each. These layers follow the appropriate shape (rectangular, triangular or parabolic) of the original layer, and if $rk = o(N)$ then the additional number of extra rows required is negligible. If there were L original layers then the total number of extra columns required will be $O(LkN/r)$. Because the networks we consider in this paper have $L = O(\log^2 N)$, if we choose $r = \sqrt{N}\log N$ then the total additional area is $o(N^2)$ for any $k = o(\sqrt{N}/\log N)$.

- Finally, one could try to show upper bounds for practical instances of the problems, e.g., a bitonic sorting network with 1024 input nodes, for which a trivial layout would give an area of $2522K$. Here one may attempt to get the best combination of the layouts from Sections 3.1, 3.2 and 3.3.

References

[1] A. Avior, T. Calamoneri, S. Even, A. Litman, and A. Rosenberg. A tight layout of the butterfly network. In *Proc. of the 8th ACM Symposium on Parallel Algorithms and Architectures (SPAA '96)*, pages 170–182, 1996.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. *Proc. of the 15th ACM Symposium on Theory of Computing*, pages 1–9, Boston, Massachusetts, 1983.

[3] K. Batcher. Sorting networks and their applications. *Proc. of the AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[4] E. Biersack, C. Cotton, D. Feldmeier, A. McAuley, and W. Sincoskie. Gigabit networking research at Bellcore. *IEEE Network*, 6(2):30–40, 1992.

[5] A. C. Dusseau, D. E. Culler, K. E. Schauser, and R. P. Martin. Fast parallel sorting under LogP: Experience with the CM-5. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):791–805, 1996.

[6] K. Eng and M. Karol. Gigabit-per-second ATM packet switching with the growable switch architecture. *Proc. of IEEE Globecom 91*, 3:1075–1081, 1991.

[7] S. Even. Layout of sorting networks. *Bell Labs Technical Note*, 1997.

[8] S. Even. Layout of the sorting net in grid-area $3n^2$. *Bell Labs Technical Note*, 1997.

[9] S. Even, S. Muthukrishnan, M. Paterson and S. Sahinalp. Grid layout of the bitonic sorter. *Proc. of the 10th ACM Symposium on Parallel Algorithms and Architectures (SPAA '98)*, pages 172–181, 1998.

[10] A. Frank. Disjoint paths in a rectilinear grid. *Combinatorica*, 2:361–371, 1982.

[11] A. Frank. Packing paths, circuits, and cuts - a survey. In *B. Korte, L. Lovasz, and A. Schrijver. Paths, Flows and VLSI Layout*, Springer Verlag, 1991, 47–100.

[12] I. S. Gopal and D. Coppersmith and C. K. Wong. Optimal wiring of movable terminals. *IEEE Transactions on Computers*, C-32:845–858, 1983.

[13] A. Huang and S. Knauer. STARLITE: A wideband digital switch. *Proc. of IEEE Globecom 84*, pages 121–125, 1984.

[14] W. Kautz, K. Levitt, and A. Waksman. Cellular interconnection arrays. *IEEE Transactions on Computers*, C-17:443–451, 1968.

[15] M. Kaufmann and K. Mehlhorn. Routing problems in grid graphs. In *B. Korte, L. Lovasz, and A. Schrijver. Paths, Flows and VLSI Layout*, Springer Verlag, 1991, 165–184.

[16] J. Kneuer. Personal communication. 1998.

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

[18] T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, 34:344–354, 1985.

[19] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, San Mateo, 1992.

[20] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* B.G. Teubner, Stuttgart, 1990.

[21] K. Mehlhorn, F. Preparata, and M. Sarrafzadeh. Channel routing in knock-knee mode: Simplified algorithms and proofs. *Algorithmica*, 1(2):213–221, 1986.

[22] H. Okamura and P. Seymour. Multicommodity flows in planar graphs. *J. Combin. Theory*, 31:75–81, 1981.

[23] R. Palmer. An experimental ATM switch for BISDN studies. *International Journal of Digital and Analog Communication Systems*, 3(4):341–349, 1990.

[24] C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322–328, 1990.

[25] R. Y. Pinter. On routing two-point nets across a channel. *Proc. of the 19th Design Automation Conference*, pages 894–902, 1982.

[26] N. Pippenger. Telephone switching networks. *AMS Proc. Symposia in Applied Mathematics*, 26:101–133, 1982.

[27] C. D. Thompson. Area-time complexity for VLSI. *Proc. of the 11th ACM Symposium on Theory of Computing*, pages 81–88, 1979.

[28] C. D. Thompson. A complexity theory for VLSI. Technical Report CMU-CS-80-132, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[29] D. S. Wise. Compact layouts of banyan/FFT networks. *VLSI Systems and Computations*, pages 186–195, 1981.

[30] E. Witte. A quantitative comparison of architectures for ATM switching systems. Technical Report WUCS-91-47, Washington University, St. Louis, MO, 1991.

[31] I. Widjaja and A. Leon-Garcia. Starburst: A flexible output-buffered ATM switch with $N \log^2 N$ complexity. *Proc. of the 14th International Switching Symposium*, Vol. 2, pages 226–230, 1992.

[32] D. Wagner and K. Weihe. A linear time algorithm for edge-disjoint paths in planar graphs. *Combinatorica*, 15:135–150, 1995.