CMU-CS-80-140

A Complexity Theory for VLSI

by
C. D. Thompson
August 1980

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-80-140

A Complexity Theory for VLSI

Dissertation presented to

Carnegie-Mellon University
Computer Science Department

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

by

C. D. Thompson

August 1980

This research is supported in part by the National Science Foundation under Grant MCS 78-236-76 and a graduate fellowship, and in part by the Office of Naval Research under Contract N00014-76-C-0370.

Abstract

The established methodologies for studying computational complexity can be applied to the new problems posed by very large-scale integrated (VLSI) circuits. This thesis develops a "VLSI model of computation" and derives upper and lower bounds on the silicon area and time required to solve the problems of sorting and discrete Fourier transformation. In particular, the area A and time T taken by any VLSI chip using any algorithm to perform an N-point Fourier transform must satisfy $AT^2 \geq cN^2 \log^2 N$, for some fixed $c > 0$. A more general result for both sorting and Fourier transformation is that $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$, for any x in the range $0 \leq x \leq 1$. Also, the energy dissipated by a VLSI chip during the solution of either of these problems is at least $\Omega(N^{3/2} \log N)$. The tightness of these bounds is demonstrated by the existence of nearly optimal circuits for both sorting and Fourier transformation. The circuits based on the shuffle-exchange interconnection pattern are fast but large: $T = O(\log^2 N)$ for Fourier transformation, $T = O(\log^3 N)$ for sorting; both have area A of at most $O(N^2 / \log^{1/2} N)$. The circuits based on the mesh interconnection pattern are slow but small: $T = O(N^{1/2} \log\log N)$, $A = O(N \log^2 N)$.

Keywords: Computational complexity, information theory, graph embedding, mesh connections, shuffle-exchange connections, parallel algorithms, sorting, Fourier transformation, VLSI, area-time complexity.

Table of Contents

Chapter 1
Introduction

Very large-scale integrated (VLSI) circuit technology has profoundly changed the size and speed of computing structures. A VLSI microcomputer occupies less than a square centimeter of silicon, yet outperforms several cubic feet of twenty-year-old computer components. The circuit densities attainable with VLSI are already staggering, and further improvements lie on the horizon. Chips with one hundred thousand transistors are feasible today. This figure may well increase to ten or twenty million in the next decade [Mead 80].

The computational power of a chip is often measured by the number of transistors it contains. This is a misleading approach, for the organization of a chip's circuitry has a very strong effect on its size and speed. In general, the more regular chip designs make more efficient use of silicon area. Such designs use less area for the wiring between transistors, leaving more room for the transistors themselves. This explains why present-day technology can put one hundred thousand transistors on a memory chip but only ten thousand transistors on a "random logic" chip. It also indicates that circuit size is more naturally measured by area than by counting transistors.

This thesis explores the relation between the speed and size of VLSI circuits, using the methodology of complexity theory. The first step in this methodology is to devise an accurate and precisely-defined model of a VLSI chip. It is then possible to derive limits on the area and time performance of any chip built according to the rules of the model. This thesis proves both upper and lower bounds on VLSI chip performance. A sample lower bound is that any chip that performs an N-point

Fourier transform in time T must have an area A large enough to satisfy $AT^2 \geq cN^2 log^2 N$, for some fixed $c > 0$. The corresponding upper bound is obtained by designing a chip that can solve a Fourier transform. The designs presented in this thesis are nearly optimal in their AT^2 performance, demonstrating that there is not much room for improvement in either the upper or the lower bounds.

The use of a new model of computation in this thesis is justified by the novel aspects of VLSI design. As indicated above, the size of a VLSI chip is best measured by its area. The useful area of a chip is devoted to transistors and wires; neither can be neglected in a realistic model. Unfortunately, Turing machines and other traditional models of computation lack the concept of wire area. Yet it is precisely this concept that is used in Chapter 3 to prove lower bounds on area-time performance.

The main results of this thesis are expressed as area-time tradeoffs. The product of chip area A with the square of the time T it takes to perform an N-element Fourier transform or sorting problem must satisfy $AT^2 = \Omega(N^2 log^2 N)$. That is, $AT^2 \geq cN^2 log^2 N$ for some fixed $c > 0$ and $N > N_0$. This lower bound is nearly the best possible, in the sense that there exist both fast, large chips and slow, small chips that nearly achieve these bounds. The small chips solve their problems in time proportional to $N^{1/2} loglog N$, using area proportional to $N log^2 N$. The near-optimality of these designs is immediate, since their AT^2 performance exceeds the lower bound quoted above by a factor of only $O(loglog^2 N)$. The fast chips are also near-optimal. The Fourier transform chip operates in $O(log^2 N)$ time, while an analogous design solves a sorting problem in $O(log^3 N)$ time. Both chips occupy $O(N^2/log^{1/2} N)$ area.

A more general result bounds the performance of any chip with area A that takes time T to solve an N-element sorting or Fourier transformation problem: $AT^{2x} = \Omega(N^{1+x} log^{2x} N)$, for all x such that $0 \leq x \leq 1$. Each value of x corresponds to a utility function AT^{2x} with slightly different weights given to area

and time performance. The lower bound on AT^2 performance given above is merely a special case ($x = 1$) of this general result.

The general lower bound implies that a chip with performance $A = O(N)$ and $T = O(N^{1/2} \log N)$ would be optimal under any AT^{2x} metric, $0 \leq x \leq 1$. The slow, small chips described above come quite close to these performance figures. The fast and large chips are far from optimal in this general sense. They only approach optimality when x is nearly 1, that is, when time is much more important than area.

The following section discusses units of measurement of VLSI circuits from a physical standpoint. The product of chip area with time is shown to be a measure of energy, leading to the following corollary of the general lower bound result: at least $\Omega(N^{3/2} \log N)$ units of energy must be dissipated during the sorting or Fourier transformation of N numbers on a VLSI chip.

The remainder of the current chapter (Sections 1.2 through 1.4) is devoted to a definition of the problems of sorting and Fourier transformation, the establishment of some notational conventions, and a review of the relevant literature.

Chapter 2 develops a "VLSI model of computation" as a basis for the derivation of lower and upper bounds on chip performance. The notion of a "communication graph" is introduced as the formal analog of a VLSI chip. Communication graphs correspond to VLSI chips, according to the scheme described in this chapter.

Lower bounds are the subject of Chapter 3. A key parameter of any communication graph is defined, its "minimum bisection width." The minimum bisection width of a communication graph determines lower bounds on the area and speed of its corresponding VLSI chip. In brief, the area A of a chip is at least proportional to the square of the width of its communication graph, ω^2. The maximum-possible speed of a chip also increases with its width; more precisely, $T = \Omega(N \log N)/\omega$. Multiplying the first inequality by the square of the second gives the lower bound $AT^2 = \Omega(N^2 \log^2 N)$. With the additional assumption that the area A is at least $\Omega(N)$, the relation becomes $AT^{2x} = \Omega((N + \omega^2)(N \log N)/\omega^{2x})$.

It can be shown that choosing $\omega = \theta(N^{1/2})$ minimizes AT^{2x} for $0 \leq x \leq 1$, leading to the general lower bound $AT^{2x} = \Omega(N^{1+x}log^{2x}N)$.

Chapter 4 develops near-optimal chip designs, providing upper bounds on achievable VLSI performance. Four designs are presented: "fast" and "slow" chips for both the sorting and Fourier transformation problems. The fast chips are based on the shuffle-exchange interconnection pattern, while the slow ones are based on mesh-type connections.

1.1 Units of area, time, information, and energy

A VLSI chip may be thought of as a multi-layered, planar structure. Transistors are formed on the surface of a silicon substrate, and cannot be stacked on top of each other. Above them is one or more layers of interconnect material (often a metal) that is selectively etched away to form connections or wires between the transistors. The conductive layers are normally insulated from each other, so that wires can cross over one another if they are formed in different layers. Wire segments in different layers can be connected to each other, if a "via" or hole is formed in the insulation.

For concreteness, the area calculations of this thesis assume there is only one layer of interconnect material, and that wires are laid out on a rectangular grid. Thus wires may meet only at right angles. Wires may also cross over each other at right angles, if one of them makes a short run in a heavily doped "channel" in the silicon substrate.

The upper and lower bound results of this thesis could be extended to cover chips with multiple layers of interconnection, as indicated by the discussion on page 36 and by Theorem 3: k layers of interconnection decrease chip area by at most a factor of k^2. There is little immediate importance to such an extension, since current chips use at most two layers of interconnection. Future chips are also likely to have a small number of layers, due to manufacturing difficulties and costs. Each

additional layer of interconnection requires one or more additional masking steps, to define where the wires are to run. Every manufacturing step contributes to chip cost both directly (because of increased fabrication time) and indirectly (because of reduced yields). Thus VLSI will continue to be essentially planar until radical advances are made in fabrication techniques.

For convenience, the assumptions of the VLSI model of computation made in this thesis are numbered in order of appearance and collected in a list on page 44. (In this list, each assumption is split into two parts, labelled "L" and "U." Part "L" contains the suppositions strictly necessary for the lower bound proofs, while part "U" contains the additional suppositions needed by the upper bound constructions.) The assumptions made in the previous paragraphs may be summarized as (L1): horizontal and vertical wires are formed in a single planar layer, although two wires may cross each other at right angles.

There is a natural *unit of area* for VLSI. Manufacturing and physical limitations give rise to a minimal spacing between the centers of parallel wires. In the terminology of Mead and Conway [Mead 80], this minimal spacing is 4λ. The square of this length, $16\lambda^2$, is thus a convenient area unit. The area of a chip can be expressed in terms of unit squares, leading to (L1, extended): one unit square is just large enough to contain one small transistor or one wire cross-over, and just wide enough to allow one wire to enter through each (unit-length) edge. The 64K RAM currently available has an area of about $10^6\lambda^2$, and chips of 10^8 or $10^9\lambda^2$ may be possible [Mead 79].

The total area of a VLSI chip may be evaluated in two ways. In production, it is the mask size that is important, that is, the area of the smallest bounding rectangle. This is another important assumption, (U2): the area of a bounding rectangle is used to describe the upper bounds (circuit constructions) of this thesis. On the other hand, the lower bounds derived here measure only the area actually occupied by wires. This is assumption (L2). It strengthens the lower bound results and allows the derivation of bounds on energy dissipation, as noted later in this section.

Information is measured in bits. One bit of information describes the outcome of an event for which there were two equally likely possibilities. For example, a one-bit signal can communicate the result of flipping a fair coin (heads or tails). More generally, the information content of an event that has outcome i with probability p_i is $\Sigma(-p_i \log p_i)$ bits [Shannon 49]. This quantity is also called the entropy of the probability distribution P.

The unit of *time* is defined by (L3): a unit-width wire has at most unit bandwidth. In other words, a signal that encodes one bit of information has a duration of at least one time unit. For binary logic, a lower bound on the length of the time unit is the duration of the shortest pulse that can change the state of a circuit.

The definitions of area, time, and information given above are chosen to simplify the theoretical model. Their values in actual implementations will depend on engineering decisions. For example, the unit of time will probably be stretched by a factor of ten or more to allow for propagation delay and synchronization overhead. In any event, the asymptotic results of this thesis will be valid to within a constant factor as long as the definitions of area and time remain fixed. These units of measurement must not change with the size of the circuit being built or with the size N of the problem being solved.

Unit values for area and time in a currently feasible MOS technology are *10* μm and *50* ns, respectively [Mead 80]. In other words, wires are at *10* μm spacings and the clock signal used for synchronization has a period of *50* ns. The units of AT^2 in this case are *250000* $\mu m^2 ns^2$. Anticipated advances in technology will reduce unit widths and times by a factor of *10*, bringing the AT^2 unit to *25* $\mu m^2 ns^2$.

A unit of *energy* is defined by the product of the units of area and time. When a signal is sent from one transistor to another, the driver must charge (or discharge) the capacitance presented by both the wire and the receiver. The energy required to charge a capacitor is proportional to its capacitance, so that the energy consumed by a wire or a transistor is proportional to its area (the constant of proportionality

depends on the way in which the chip is fabricated). Thus it can be said that one unit of energy is consumed by one unit of chip area every time it is involved in the transmission of a signal.

The proofs of Chapter 3 place a lower bound on the amount of wire that must be active continuously if a circuit is to compute a function in a given time T. The general lower bound on AT^{2x} performance can therefore be used at $x = \frac{1}{2}$ to give a lower bound on the total energy required to compute a function: $AT = \Omega(N^{3/2} \log N)$, for both sorting and Fourier transformation.

The unit of energy for a MOS chip can be evaluated in the following way. Currently, wires have about 10^{-3} pF of capacitance per unit area ($100\ \mu m^2$). Assuming that the signal voltage swing on a wire is 4 volts, .008 pJ of energy ($= \frac{1}{2}CV^2$) is needed to charge each unit of wire area, each time its voltage is changed. Thus the product of wire area with the amount of time it is actively transmitting data has the dimensions of energy, and units of .008 pJ. This energy unit will be reduced to 8×10^{-8} pJ, when lengths, times, and voltages are scaled down by the predicted factor of 10 [Mead 80].

1.2 Problem definitions

The two computational problems treated in this thesis are functions of N variables onto N variables. A VLSI chip is said to solve one of these problems if it can produce the appropriate N output values from any vector of N input values. Input and output values remain on the chip. This assumption is in accordance with a paradigm of comparing a VLSI chip with a conventional computer. Just as complexity issues can be studied without reference to the I/O devices on a conventional computer, there is no need to model off-chip communication if there is a very large amount of memory on the chip. (The assumption of on-chip computation is implicit in assumptions L6 and L7, as defined in Section 2.1.)

The values for problem variables are chosen from the elements of a finite set.

That is, the value of a variable may be encoded as a "word" composed of a bounded number of bits. If there are no more than M possible values for each variable, then a word length of $\lceil \log M \rceil$ bits can be used.

A particular assignment of values to input variables constitutes a "problem instance." These values must be chosen independently and uniformly, leading to (L4): there are M^N equally likely problem instances, if each of the N input variables can take on M different values. Assumption L4 is more powerful than it may appear at first glance. Its insistence on an independent, uniform distribution of input values allows many algebraic simplifications in the derivation of time bounds, as will be seen in Section 3.3.

A chip is said to solve a problem in *average* time T if it takes an average of T units of time to solve one of the M^N equally likely problem instances. In a similar fashion, a chip is said to solve a problem in *worst-case* time T if it takes no more than T units of time to solve any instance of the problem.

For lower bounds, an average-case result is strictly stronger than an equivalent worst-case result. A chip with an average-case time of at least T must also have a worst-case time of at least T. The lower bound on Fourier transformation covers the average case, while the sorting lower bound applies only to the worst case. Obtaining a lower bound on achievable performance for the average case of a sorting chip remains an open problem.

The upper bound results of this thesis use nonadaptive (straight-line, non-branching) algorithms, so that the chips operate at the same speed on any problem instance. The best-, average-, and worst-case performances of the chips are thus identical.

1.2.1 Discrete Fourier transformation

The central problem studied in this thesis is the computation of the discrete Fourier transform, or DFT. The DFT may be defined as a matrix-vector

multiplication, $A\tilde{x} = \tilde{y}$, where A is the matrix of constants defined below. The input vector is $\tilde{x}$ and the output vector is $\tilde{y}$; both are of length N. The elements of the N-by-N matrix A are constants determined by the structure of the ring in which the computation is performed.

A ring is defined by a set of elements and the rules for adding and multiplying elements. The values of variables in $\tilde{x}$ and $\tilde{y}$ must be chosen from the elements in the ring. For the reason noted in the previous section, there can be only a finite number of values for each of the problem variables. The customary definition of a Fourier transform over the (infinite set) of complex reals is thus inadmissable here; only finite rings will be considered in this thesis.

The structure of the ring over which the DFT is defined has some impact on the properties of the transform. It is customary [Agarwal 75, Aho 74, Bonneau 73, Nicholson 71] to restrict attention to commutative rings with additive and multiplicative identity elements, a principal Nth root of unity, and a multiplicative inverse for N. Such rings lead to DFTs with most of the properties associated with Fourier transforms in the field of complex numbers: invertibility, orthogonality, and the cyclic convolution property [Agarwal 75].

A particularly suitable ring for the DFT is the integers $\{0, 1, \ldots, M-1\}$ under addition and multiplication modulo M. The prime factorization of the modulus M characterizes this ring. If

$$M = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k}, \qquad (1.1)$$

then there is an Nth root of unity and an element N^{-1} if and only if N divides the greatest common divisor of $\{p_1 - 1, p_2 - 1, \ldots, p_k - 1\}$ [Agarwal 75, Bonneau 73]. An immediate implication of this result is that $M > N$.

If the constant α is a principal Nth root of unity, the matrix A in the defining equation $\tilde{y} = A\tilde{x}$ is given by

$$A[i,j] = \alpha^{ij}, \text{ for } 0 \leq i,j \leq N. \qquad (1.2)$$

The elements $A[i,1]$ are all distinct [Agarwal 75].

The length of the transform, N, determines what algorithms may be used to compute a DFT. Winograd's [Winograd 76] DFT algorithm is based on a recognition of this effect. The upper bounds of Chapter 4 are implementations of the "fast Fourier transform" or FFT [Aho 74], and thus assume (U4): N is a power of 2. In contrast, the lower bounds of Chapter 3 apply to all N.

The Fourier transform circuits of Chapter 4 use an additional assumption, also part of (U4): $\log M = O(\log N)$. This assumption allows the value of any input or output variable to be coded in $O(\log N)$ bits, so that the dependence of circuit size on M can be expressed in terms of N alone. An interesting existence question is raised by this assumption. For arbitrary N, there had better be a ring modulus M of appropriate size capable of supporting an N-element DFT. According to Agarwal's result, cited above, an N-element DFT exists in a ring of prime modulus M iff M is of the form $qN+1$. Fortunately, it can be shown [Wagstaff 79, Linnik 44] that the least prime M of this form is bounded by a polynomial in N. A word-length of $\log M = O(\log N)$ bits is thus available for any N-element DFT. Assumption U4 merely states that the circuit constructions of Chapter 4 use nearly minimal word-lengths.

1.2.2 Sorting

The N inputs to a sorting chip may be thought of as integers between 0 and $M-1$. This analogy to the integers is intended to convey two things. First, the input values are members of a linearly ordered set, so that a "greater than" relation is defined. Second, there are exactly M different possibilities for each input value.

The N outputs of a sorting chip are a permutation of the inputs into sorted order. That is, output y_0 is the smallest input value, output y_1 is the second smallest, and so forth.

The lower bound of Theorem 15 requires an addition to assumption (L4): $M = N^{1+\varepsilon}$ for some fixed positive ε. Note that some lower limit must be assumed

for M in order to obtain a powerful result. For example, the sorting problem for $M = 1$ is trivial; no computation is required to determine that all output values are *0*.

1.3 Notation

The following functional notation is used throughout this work.

$f(n) = O(g(n))$	"f is big O of g," an upper bound within a constant factor. There exists a positive constant c for which $f(n) \leq cg(n)$ for all sufficiently large n.
$f(n) = \theta(g(n))$	"f is theta of g," an exact bound within a constant factor. There exist positive constants c_1 and c_2 for which $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all sufficiently large n.
$f(n) = \Omega(g(n))$	"f is omega of g," a lower bound within a constant factor. There exists a positive constant c for which $f(n) \geq cg(n)$ for all sufficiently large n.
$\lceil x \rceil$	"ceiling of x," the least integer greater than or equal to x.
$\lfloor x \rfloor$	"floor of x," the greatest integer less than or equal to x.
$\log x$	the base two logarithm of x.
$\log^y x$	$(\log x)^y$.
$\log\log^y x$	$(\log\log x)^y$.
$\|X\|$	the number of elements in the set X or dimensions in the vector X.
$\|\{X\}\|$	the number of distinct values in the sample space of the (discrete) random variable X.

1.4 Related work

Three existing areas of inquiry shed light on the problems studied here. First, applications of a theory of *graph embedding* in the plane, such as printed circuit board wire routing, are relevant to the essentially planar VLSI technology. Second,

the model of computation developed here for VLSI is similar in spirit to other *information theoretic models.* Third, the area-time results derived in this thesis may be contrasted with the *space-time tradeoffs* observed in more traditional models of computation.

Graph embedding. Results in this area are written in a wide range of styles, from a hard-nosed pragmatic approach to a carefully-formalized theoretical treatment.

On the practical end, the problem of wire and chip placement on printed circuit boards is quite similar to the problem of wire and circuit placement on the surface of VLSI chips. Donath [Donath 79] derives upper bounds for wire length in both problem domains, as a function of a parameter in the empirical "Rent's relationship" for logic networks. These upper bounds overestimate wire lengths by a factor of two or less, on a number of actual layouts. Sutherland and Oestreicher [Sutherland 73] estimate wiring requirements for printed circuit boards, under the pessimistic assumption that chips are randomly placed on the board. Both studies use the key idea, developed here in Section 3.1, of obtaining analytical results by partitioning the graph.

Formal approaches to graph embedding also yield interesting results. Cutler and Shiloach [Cutler 78] study the problem of embedding bipartite graphs in the plane, under the rather restrictive assumption that no edge "crossovers" are allowed. Lipton and Tarjan [Lipton 77] and Rosenberg [Rosenberg 79] obtain bounds on the cost of embeddings, under the very liberal assumption (for VLSI) that any number of edges may pass over a point. Leiserson [Leiserson 80] uses more natural assumptions in his algorithm for embedding a graph in near-minimal area. The last three papers cited use variants of the partitioning idea alluded to in the previous paragraph. Lipton and Tarjan take the idea one step further, and use the size of the partition to derive complexity results. Much of their work should transfer into the VLSI model of computation, but this task has not yet been attempted.

Information-theoretic models. An information-theoretic model is defined here as

one that emphasizes the cost of information transmission. Most existing models have a different orientation, measuring operation counts and memory requirements for the traditional von Neumann architecture for uniprocessors. Even so, the informational emphasis of the VLSI model of computation is far from unique.

Among established models in complexity theory, cellular automata [von Neumann 66] are the most suited for informational studies. Each cell of a two-dimensional array of automata changes its state as a function of the current states of its nearest neighbors. From an information-theoretic standpoint, each cell receives a packet of information from its neighbors every time unit. This packet need have no more bits than the logarithm of the number of possible neighborhood state configurations. Minimal time and state solutions to cellular automata problems thus tend to minimize the transmission of information. Moshell and Rothstein's "bus automata" could be used to model the flow of information among cellular automata in a natural fashion [Moshell 76].

Floyd [Floyd 72] makes use of the entropy of a memory state to obtain lower bounds on the number of operations needed to perform memory reorganization in a two-level store. In his model, an operation is the production of a third "page" of information, as any function of the contents of any two pages. If n_{ij} is the number of records (amount of information) to be sent from the *ith* page to the *jth* page, the entropy of a memory state is defined as $\Sigma\, n_{ij} \log n_{ij}$. (Actually, Floyd's V-function is defined somewhat differently to handle the case that $\log n_{ij}$ is not an integer.) One operation can change the entropy of a memory state by at most p, where p is the number of records on a page. A lower bound on the number of operations needed to achieve a reorganization is thus the entropy of the original state (relative to that reorganization) divided by p.

On another front, a theory of "distributed computing" is beginning to emerge as an outgrowth of research into parallel processing for database manipulation. A. Yao [Yao 79] outlines some of the implications of various assumptions that might be made about a distributed system: one-way *vs.* two-way communication,

deterministic *vs.* probabilistic computations. Abelson [Abelson 80a] develops some analytic tools for bounding the information transfer required in the computation of continuous, differentiable functions. Both authors treat a problem as a fixed partitioning of the input data between a pair of processors. In distinction, this thesis defines a problem as an input-output relation, with no partitioning assumptions.

Space-time tradeoffs. Although operation counts and memory requirements of uniprocessors are irrelevant considerations for VLSI, the analytical tools to demonstrate space-time tradeoffs in more conventional models can be applied to VLSI area-time studies.

Grigoryev [Grigoryev 76] studies the problem of computing a set of m binary functions. He proves that any straight-line (nonbranching, nonadaptive) algorithm with T steps and S storage locations satisfies $ST > ml/2$, if the set of functions is "l-independent." A similar notion of functional independence is the basis for the bounds of Section 3.3. Grigoryev's definition is discussed in more detail in that section. Savage and Swamy [Savage 79a] generalize Grigoryev's method and apply it to integer multiplication. Earlier, they had found a space-time tradeoff for the fast Fourier transform algorithm [Savage 77].

Space-time bounds are often derived from consideration of a "pebble game" [Paterson 70, Savage 77] played on the graph of a straight-line algorithm. Each pebble corresponds to a storage register and each node represents a function to be evaluated. An edge leads from one node to another if the value of the parent appears as a parameter in the child's function. Nodes with no parents correspond to problem inputs; nodes with no children are problem outputs. Placing a pebble on a node means storing the value of the node's function in the pebble's register, which is possible only if the node's parents (the function's parameters) are all pebbled (evaluated and stored). Removing a pebble from a node corresponds to erasing the contents of the pebble's register. The object of the game is to pebble all childless nodes, in other words, to evaluate all the problem outputs. Time in this model is measured serially as the number of pebble movements, that is, the number of function evaluations. Space is the number of different pebbles (registers) used.

For VLSI, one is tempted to interpret the graph of an algorithm in quite a different fashion. Each node could be a processor waiting for its predecessors to send it the results of their function evaluations. Time would then be the depth of the graph, and space the area of the graph when embedded in the plane. The pebbling game seems irrelevant in this interpretation of a graph. Some pebbling results, however, involve proofs of necessary properties of any graph for a particular problem. For example, Valiant [Valiant 76] shows that any Fourier transform algorithm corresponds to a hyperconcentrator. Pippenger's extension of this result, as reported by Tompa [Tompa 78], is the basis of Lemma 8 of Section 3.3.1.

Unfortunately, most pebbling results are based on algorithms operating over the set of real numbers. The analytic techniques used do not always transfer into the modular arithmetic, or other finite algebraic structures, of the VLSI model of computation.

Chapter 2
The VLSI model of computation

A VLSI chip composed of transistors and interconnections is modeled by a network of *nodes* and *wires*. A node represents a transistor or a small cluster of transistors; as such, it receives and transmits signals over its connecting wires. A node may also represent a wire junction, in which case it merely copies the signals it receives on any one of its wires onto its other wires. Nodes and wires thus simulate the actions of transistors and wires on a VLSI chip.

Nodes are capable of storing a limited amount of information. This enables them to model data storage elements on a VLSI chip. It also allows a collection of nodes and wires to be a complete, self-contained computing structure. The inputs to a computation are stored in a distinguished set of nodes called *source nodes*. (These correspond to the "input registers" on a VLSI chip.) The output values of a computation are collected in another set of nodes called *sink nodes* (the "output registers" on a chip). A collection of nodes and wires capable of solving a problem is called a *communication graph*. A communication graph is thus the formal analog of a VLSI chip.

Section 2.1 contains precise definitions of the functional capabilities of nodes and wires, and the ways in which they may be put together to form communication graphs. The lower bound proofs of Chapter 3 (which apply to communication graphs) demonstrate that these definitions imply limits on the area and time performance of communication graphs.

Every VLSI chip can be accurately modeled by a communication graph, as shown

by a correspondence scheme described in Section 2.2. For this reason, the lower bound results of Chapter 3 are valid for all VLSI chips as well as for all communication graphs.

In Section 2.3 we turn to issues of upper bounds. An upper bound in the VLSI model of computation implies the existence of a chip achieving the stated performance. Unfortunately, not all communication graphs defined in Section 2.1 correspond to feasible chip designs. Section 2.3 remedies this difficulty by adding further constraints to the VLSI model of computation. Any communication graph satisfying the additional constraints of Section 2.3 is called an *admissible communication graph*. A generalized MOS process [Mead 80] may be used to implement a feasible chip based on any given admissible communication graph.

Section 2.4 discusses the relationship of the VLSI model of computation with the similar model implicit in the work of Mead and Rem [Mead 79].

The methodology of this chapter is summarized by the Venn diagram of Figure 2-1. The universe being studied is that of "all computational structures" that fit in area A and solve an N-element problem P in time T. A (possibly empty) set of communication graphs achieving this area-time performance may be constructed in accordance with the definitions of Section 2.1. This set is denoted as "(A,T,P,N)-communication graphs."

The correspondence scheme of Section 2.2 constructively demonstrates that a communication graph can be obtained from any VLSI chip. Thus the set of "(A,T,P,N)-VLSI chips" (actually, the set of (A,T,P,N)-communication graphs corresponding to VLSI chips) is a subset of all (A,T,P,N)-communication graphs.

The generalized MOS process adopted for upper bound proofs is of course only one way of building VLSI chips, so that "(A,T,P,N)-MOS chips" is a proper subset of "(A,T,P,N)-VLSI chips." Finally, the class of "admissible (A,T,P,N)-communication graphs" defined by Section 2.3 form a subset of "(A,T,P,N)-MOS chips," according to the correspondence scheme of that section.

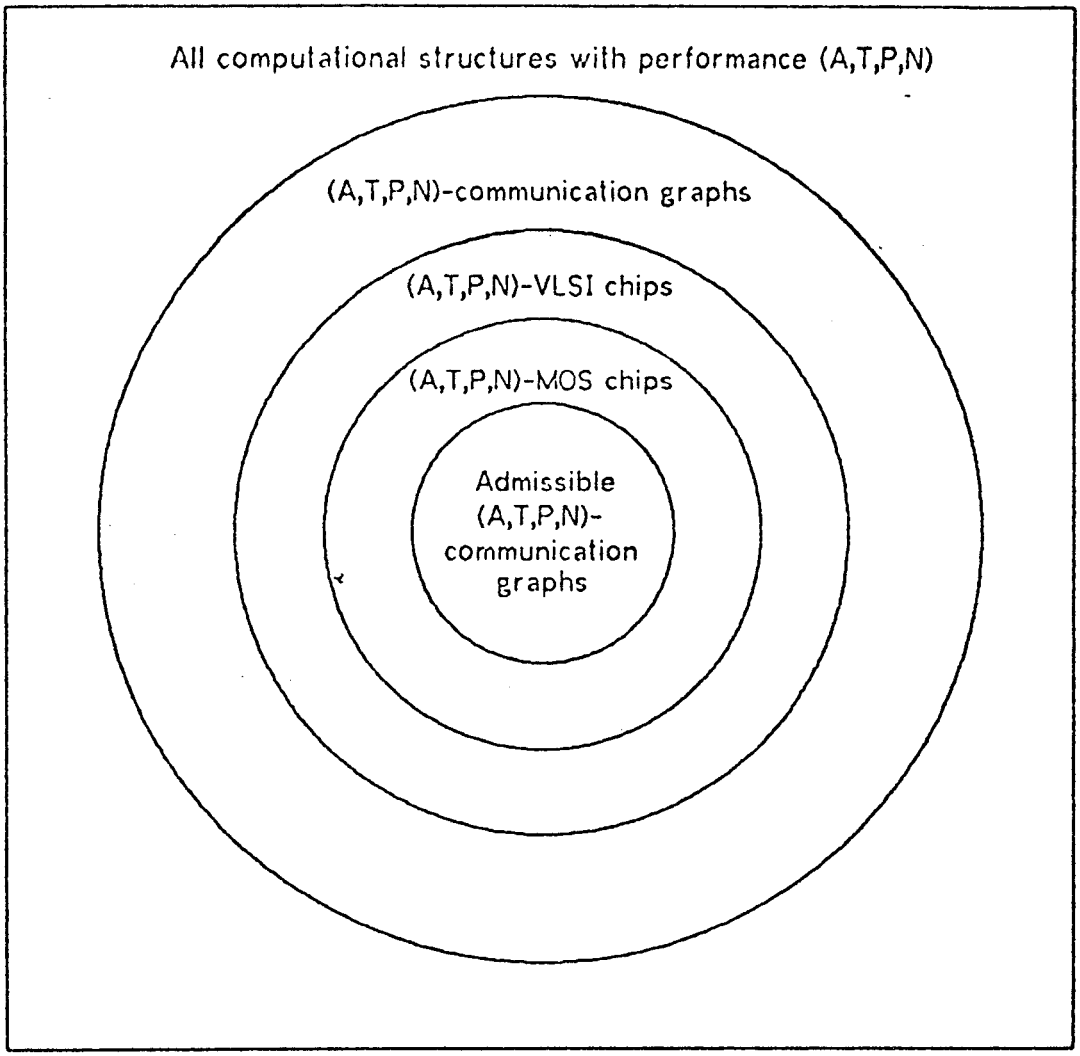

Figure 2-1: Domains of the lower and upper bound models.

This methodology illuminates the type of arguments needed to demonstrate the consistency and utility of the VLSI model of computation. For example, assumption A2 of Section 1.1 defines two measures of the area occupied by a circuit, depending upon whether the area figure is to be used as a lower or upper bound. The lower bound area is just the amount of area occupied by wires, while the upper bound area is that of the smallest bounding rectangle. This dual definition is consistent with the inclusion of "admissible (A,T,P,N)-communication graphs" in "(A,T,P,N)-communication graphs" since any graph bounded by a rectangle of area A must also have fewer than A units of wiring.

2.1 Lower bounds

A communication graph is composed of *nodes* and *wires* laid out on a grid of unit squares. Physically, a wire is a horizontal or vertical strip of metal connecting two points on the surface of a chip. A node represents a point at which wires meet, so that a transistor or even a wire junction is a node.

Assumption L1, below, defines the concept of area for the lower bound model of VLSI computation. It is a restatement of the unit-area definition given in Chapter 1 on page 7. (Some of the assumptions developed in Chapter 1 applied to lower bounds, some only to upper bounds. Accordingly, each assumption was prefixed with either an "L" or a "U" to indicate its application to lower or upper bounds. Assumptions L1 through L8 are defined in this section, forming a complete definition of the lower bound model of computation. Section 2.3 defines the upper bound model as a set of additional restrictions on these assumptions, labeled U1 through U8.)

> **Assumption L1**: Area. A unit square can contain one node or one wire cross-over. One wire may cross each edge of a unit square, so that nodes have a maximum of four wires.

There are thus nine types of squares occupied by wires, as shown in Figure.2-2. A square may also contain a node, as indicated in the tenth "tile" of this figure. An arrowhead is drawn at the point where a wire meets a node if information flows into the node from that wire. This notion of information flow will be formalized later.

Figure 2-2: Wire segments.

Assumption L2 of Section 1.1 (page 7) makes the following definition of the total area of a communication graph. Note that only occupied squares are counted toward the lower bound on area.

Assumption L2: Total area. The total area of a communication graph is equal to the number of unit squares occupied by wires or nodes.

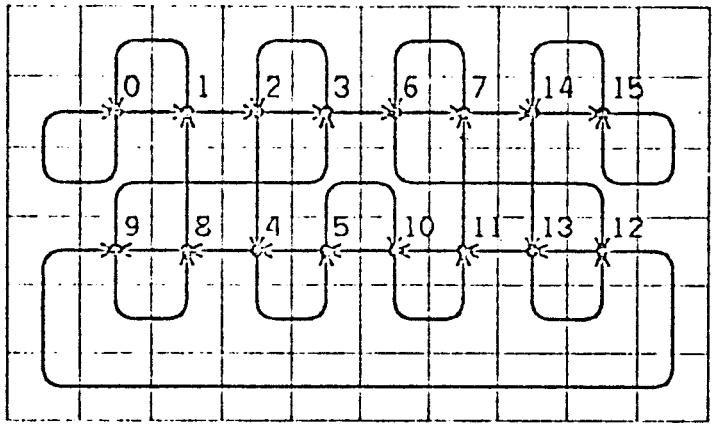

Figure 2-3: The shuffle-exchange graph of sixteen nodes, embedded in 58 unit squares (and bounded by a 60-unit rectangle).

The function of a wire is to carry information from node to node. Most wires carry information in one direction only. Such wires are drawn as a (possibly curved) line connecting two nodes, with an arrowhead pointing in the direction of information flow. The rarer bidirectional wire (such as the one in Figure 2-10) is drawn with two arrowheads.

Example. Figure 2-3 shows an embedding of the shuffle-exchange graph of sixteen nodes. The shuffle-exchange connectivity is natural for sorting and Fourier transformation [Stone 71]. A shuffle-exchange graph of size $N = 2^N$ has nodes numbered from 0 to $N-1$. Node i can transmit information to node $(2i + \lfloor 2i/N \rfloor) \bmod N$ over a "shuffle" connection. "Exchange" connections exist in both directions between nodes $2i$ and $2i+1$. Section 4.3.4 treats the general problem of embedding a shuffle-exchange graph of size N.

The following assumption bounds the rate at which one-bit signals can pass any point on a wire. This bandwidth limitation defines the unit of time for a communication graph, as indicated in assumption L3.

Assumption L3: Units of time. A wire has at most unit bandwidth in each direction.

Time bounds are obtained in the VLSI model of computation from arguments

based on the bandwidth limitation of wires. For this reason, it is important that a problem be specified in a fixed number of bits. As defined in Section 1.2, a "problem instance" is an assignment of values to input variables. By the following assumption, based on the discussion on page 10, each problem instance cannot be coded in fewer than $N \log N$ bits. (Strictly speaking, assumption L4 is not a definition of the model so much as a description of the computational problems treated in this thesis.)

> **Assumption L4**: Problem definition. Each of N input variables takes on one of M different values, for a total of M^N equally likely problem instances. In the sorting problem, $M = N^{1+\varepsilon}$ for some fixed positive ε. In the Fourier transform problem, $M > N$.

It is now possible to describe the functionality of nodes and wires. At each instant of time t the signal available at one end of wire A is expressed by the boolean variable $A(t)$. (Two boolean variables are associated with each bidirectional wire, to denote the possibly different signals available at either end.)

The value of each signal is determined by the *transmission function* of the node that originally placed it onto its wire. The simplest transmission functions describe the operations of nodes with no "state" (local storage, memory). For example, a memoryless node that has three incoming wires (A, B, and C) and one outgoing wire (D) has a transmission function of the form

$$D(t+\delta_D) = f(A(t), B(t), C(t)). \qquad (2.1)$$

The boolean variables $A(t)$, $B(t)$, and $C(t)$ denote the signals on the incoming wires at time t. The signal on wire D appears at the far end of that wire after some fixed delay $\delta_D > 0$. The function f is any of the 256 boolean functions of three boolean variables. Note that there is no explicit delay associated with the computation of f. Any such "node delay" is added to the delays of its wires. Also note that this model allows wires to act as transmission lines: wire D may have unit bandwidth even if its delay δ_D is greater than unity, for there is nothing to keep a node from transmitting another signal before the first one has been received. (No VLSI technology allows transmission lines as yet, but the VLSI model of computation is ready for them

should they become feasible. The motivation for the development of on-chip transmission lines is strong, due to the ever-increasing ratio of propagation delay to device switching time.)

Nodes with state can have more complicated transmission functions, as defined in Assumption L5.

> **Assumption L5**: Transmission functions. Node states and wire signals are completely and consistently described by the transmission function associated with each node. A node with state vector S, input wires $(A, \ldots, D)$, and output wires $(E, \ldots, G)$ computes a function of the form
>
> $$[S(t+1), E(t+\delta_E), \ldots, G(t+\delta_G)] = F[S(t), A(t), \ldots, D(t)], \qquad (2.2)$$
>
> where δ_E and δ_G are the non-negative delays of wires E and G.

There are two extremely important assumptions buried in the formalism of equation (2.2). First, the function F is defined on the instantaneous values of its variables: there is no allowance for timing "jitter" or any other synchronization difficulties. Thus the lower bound model of computation assumes that some form of synchronization between nodes is available, for which no area or time charges are made. The subject of synchronization will be discussed in more detail in Section 2.3, as the upper bound model of computation is developed.

A second buried assumption is that of determinism, that the signals coming into each node take on one of two values. Marginal and erroneous signals will appear in any real system, although careful design practice will reduce the probability of error to nearly zero. In any event, the VLSI model of computation treats only an ideal world in which there are no transmission errors.

A communication graph is not completely specified without a description of the initial states of all nodes and wires. These states must all be constants, that is, independent of the values of problem input variables, with the exception of one node for each input variable, its *source node*. In other words, all information about the value of each input variable is initially concentrated at one point. As the computation proceeds, this information will of course be diffused throughout the

communication graph. Note that the correspondence between source nodes and input variables is one-to-one. This assumption is crucial to the proofs of Section 3.3, although the recent work of Brent and Kung [Brent 79] indicates that it might be relaxed.

> **Assumption L6**: Source nodes. The initial state of a source node may be any function of the value of its input variable. Each input variable affects only the initial state of its source node.

Just as there is one source node for each input variable, there is one *sink node* for each output variable. However, sink nodes need not be in one-to-one correspondence with their variables. The function of a sink node is to collect information about the correct values for its output variables. The computation is complete when each sink node has completely determined the values of its output variables. To ensure that a sink node is not just "guessing" the right answer momentarily, it is required to come to a stable decision, in the sense formalized below.

> **Assumption L7**: Sink nodes. There is a fixed assertion for each sink node, relating its state to the correct values of its output variables (as a function of the values of the input variables). A computation is complete at time T if all assertions are satisfied at all times $t \geq T$.

The final assumption of the lower bound model defines what it means to say that a chip solves a problem: it must be able to solve all instances of that problem (assignments of values to input variables).

> **Assumption L8**: Solution time. A communication graph is said to solve a problem in worst-case time T if it takes no longer than T units of time to complete its computation of any problem instance. A communication graph is said to solve a problem in average time T if its average completion time, over all problem instances, is T.

Some additional naming and drawing conventions will prove useful when dealing with communication graphs. Nodes that are neither sources or sinks are called *switching nodes*, since they can be considered as mere "switches" or combiners of information. This classification of nodes as sources, switches, or sinks for information is not disjoint; a single node may serve in any or all of these capacities.

The following conventions are adopted for drawing communication graphs. Nodes are numbered and wires are named with capital letters. The delay of a wire is either zero or unity, and all wires emanating from a node have the same delay. In the notation of assumption L5, all the $\delta_E, \ldots, \delta_G$ for a given node have the same value, either *0* or *1*. Nodes with zero-delay wires are drawn as dots, while nodes with unit-delay wires are drawn as open circles. The logical content of node transmission functions are indicated by boolean equations on each wire.

Example. Figure 2-4 is a communication graph for a VLSI chip that computes the 'and' and 'or' functions of two boolean variables p and q. Nodes *1* and *2* are sources, node *3* is a switch, and the sink nodes are nodes *4* and *5*. Wire *A* carries information about input *p* from node *1* to node *3*. Similarly, wire *B* carries the value of input *q*. Node *3* computes the values of the two outputs, $p \wedge q$ and $p \vee q$. Wires *C* and *D* carry these output values to sink nodes *4* and *5*.

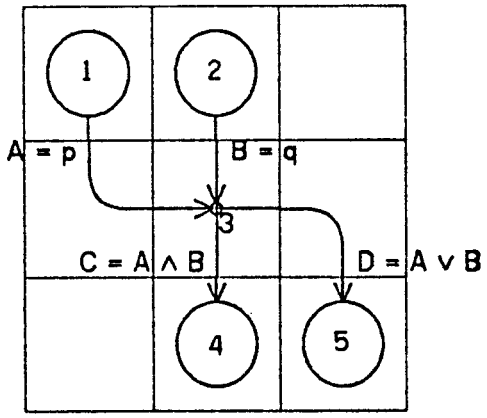

Figure 2-4: A simple communication graph.

A more precise description of Figure 2-4 is contained in the following transmission functions, initial assignments, and output assertions.

f_1: $S_1(t+1)=S_1(t)$; $A(t+1)=S_1(t)$; $S_1(0)=p$; $A(0)=false$

f_2: $S_2(t+1)=S_2(t)$; $B(t+1)=S_2(t)$; $S_2(0)=q$; $B(0)=false$

f_3: $C(t)=A(t)\wedge B(t)$; $D(t)=A(t)\vee B(t)$; $C(0)=D(0)=false$

f_4: $S_4(t+1)=C(t)$; $S_4(0)=false$; assert $S_4(t)=p\wedge q$

f_5: $S_5(t+1)=D(t)$; $S_5(0)=false$; assert $S_5(t)=p\vee q$

From this description, it may be seen that the communication graph takes two units of time to solve its problem. The initial state $S_1(0)$ of

source node *1* is defined as the value of its input variable *p*. Wire *A* carries this value during the first time unit, since $A(1)=S_1(0)=p$. In the same fashion, wire *B* carries the value of *q* during time $t=1$. Referring now to f_3, wire *C* carries $p \wedge q$ and wire *D* carries $p \vee q$ during this same period of time. (Node *3* is defined to have zero delay, so that these signals are available immediately.) The signals on wires *C* and *D* are stored in the states of nodes *4* and *5* during time $t=2$, as indicated by the first equations in f_4 and f_5. The output assertions will be satisfied for all times $t \geq 2$, so that the computation is complete after two time units.

Note that nodes *1, 2, 4* and *5* have unit delay and that node *3* has zero delay. This might be an appropriate model for a circuit that clocks its inputs at time $t=1$ and clocks its output latches at $t=2$. The next section discusses the subject of appropriate models in more detail.

The delays of nodes *4* and *5* can be reduced without violating the lower bound model of computation. However, the delays of nodes *1* and *2* must remain, to avoid sending two signals down wires *A* and *B* at the same time. (The initial value on wire *A* is *false*; a zero-delay transmission function for node *1* would also require $A(0)=p$.) Thus the computation of the 'and' and 'or' functions of two variables can be done in as little as one time unit, in the VLSI model of computation.

2.2 Correspondence to VLSI chips

The VLSI model of computation is designed so that VLSI chips correspond directly to communication graphs. This section details the way in which a communication graph is derived from any chip layout. The correspondences described here are necessarily vague, as the VLSI model of computation is intended to apply to any technology. Examples are taken from both the generalized MOS technology described by Mead and Conway [Mead 80], and from a "scaled" I^2L technology [Evans 79].

For chips with a single layer of interconnection material, the correspondence is simple. Conductive paths on the surface of the chip are modeled as wires. Wire junctions and transistors are nodes. The topologies of the domains are nearly identical. The planar silicon substrate becomes the grid of unit squares of the lower

bound model. Chips with multiple layers of interconnection material are modeled by the technique shown on page 36.

Conductors, wires. Any VLSI chip must have an abstract operational description in terms of sequences of logical values that are carried along signal paths. Such a description exists at least in the chip designer's mind, when reasoning about the ways in which data is represented, modified, and moved.

A signal path is any conductor that serves to move data. Such conductors are modeled as wires in the VLSI model of computation. The data is modeled as a sequence of binary signals carried by the wires.

Transistors, wire junctions, and nodes. The wires of a communication graph have only two ends. A node is needed to model each spot on the chip where signals can fork ("fanout") or join ("fanin").

In particular, every active device or transistor on a chip is a node. In many technologies, transistors are used for logical fanin, as illustrated in Figure 2-5.

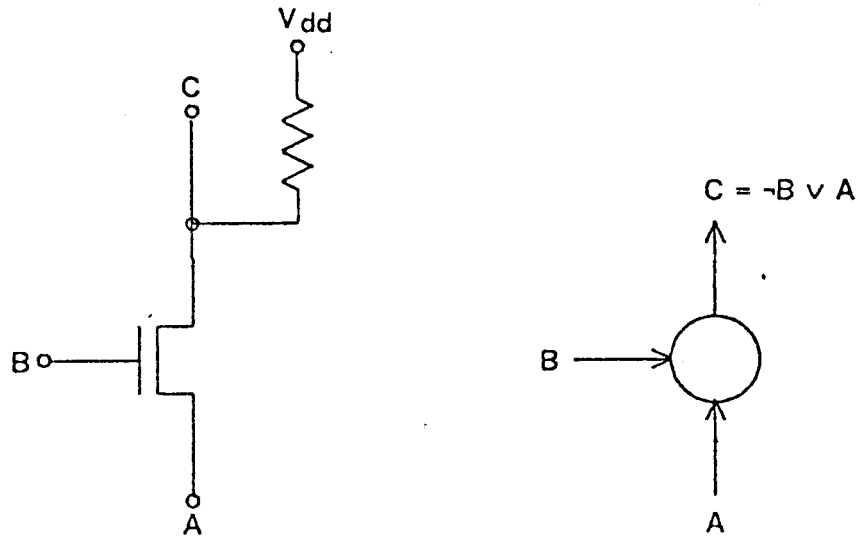

Figure 2-5: MOS circuit illustrating logic fanin at an active device, with corresponding communication graph.

The output C goes to V_{dd} (logical *1*) whenever the transistor is nonconducting (B=*0*), or whenever both A and B are at V_{dd}.

Transistors may also be the loci of logical fanout, as occurs in the I^2L transistor of Figure 2-6.

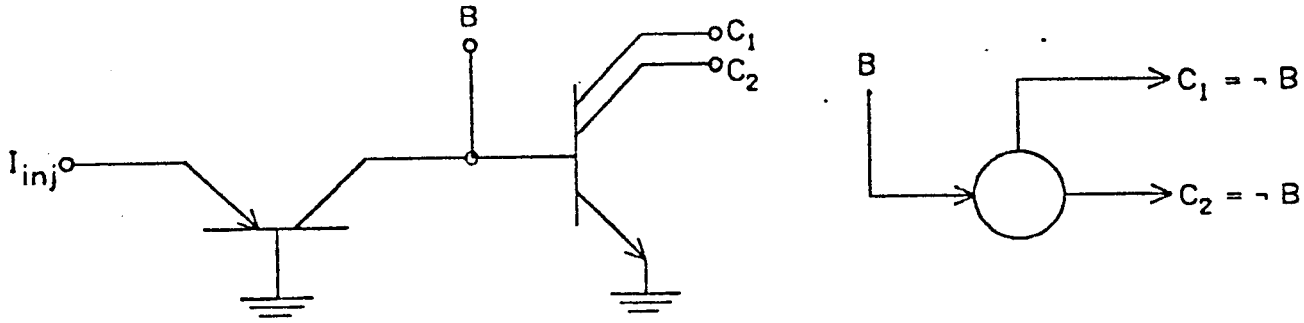

Figure 2-6: I^2L circuit illustrating logic fanout at an active device, with corresponding communication graph.

Here, a logical *0* is a high impedance state, while a wire in the *1* state is nearly shorted to ground. The outputs C_1 and C_2 are *0* if the transistor is turned off, which happens only when the injector current is short-circuited through *B*.

Fanin and fanout can also occur without the mediation of an active device. For example, a single conductor can supply its signal to many different circuits. The model for such a conductor has a number of zero-delay fanout nodes, one for each fork in the signal path, as shown in Figure 2-7.

The fact that logic fanin can occur without an active device is perhaps surprising, yet it is a fairly common design practice. The I^2L nand gate of Figure 2-8 illustrates an appropriate model for conductors that perform logic.

In the VLSI model of computation, nodes are limited to four wires, so large fanouts and fanins must be modeled by multiple nodes. This should not pose any special difficulties, as nodes are only a square wire width in size. Nodes and wires can thus model the internal communication within a large active device, as shown in Figure 2-9.

Note that nodes are allowed to have rather complex functions, by the introduction of a large vector of state bits. A state vector is required for the definition of a source or sink node, but shouldn't be needed to describe the

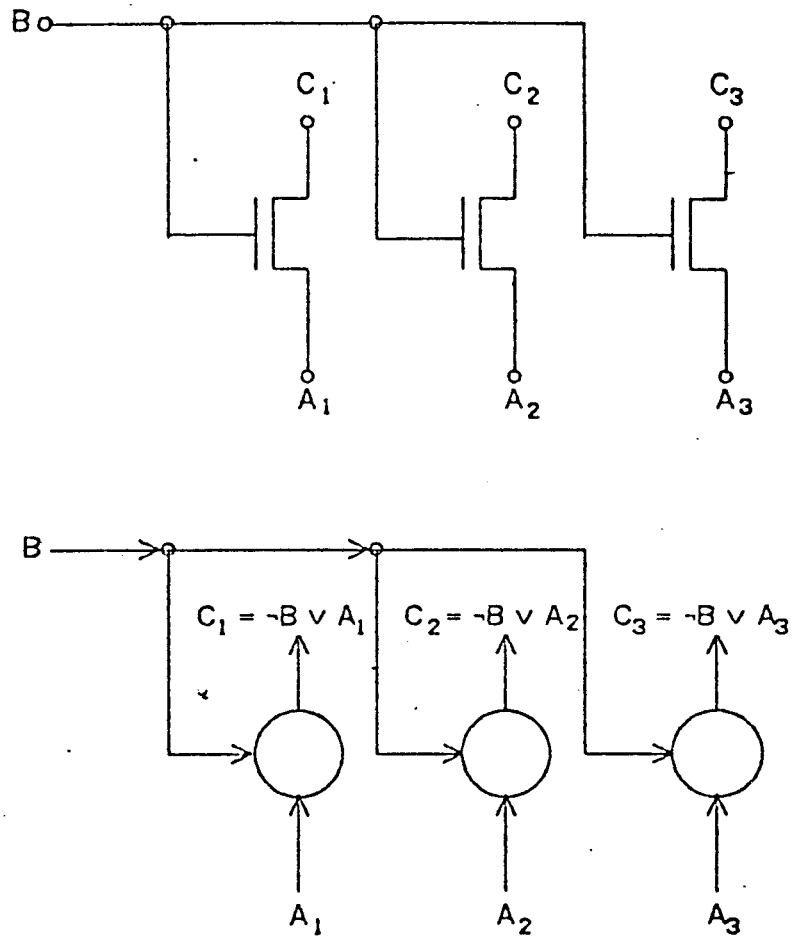

Figure 2-7: MOS circuit illustrating logic fanout on a conductor, with corresponding communication graph.

operation of any other node. An exception arises if the chip's conductors carry more than two different logical signals. Nodes work only on binary signals, so they may need a few bits of state to model circuits employing multiple-valued logic.

Source and sink node correspondence will be treated in more detail later in this section.

Power, ground, and synchronization. A large proportion of the area of any VLSI chip is occupied by conductors that distribute power and global clock pulses. No wires are drawn in a communication graph to correspond to these conductors, since they do not carry information. This omission can only strengthen the lower bound results of this thesis, which are obtained without reference to the additional area constraints imposed by such wiring. Johannsen [Johannsen 78] treats the problem of power distribution on VLSI chips in more detail.

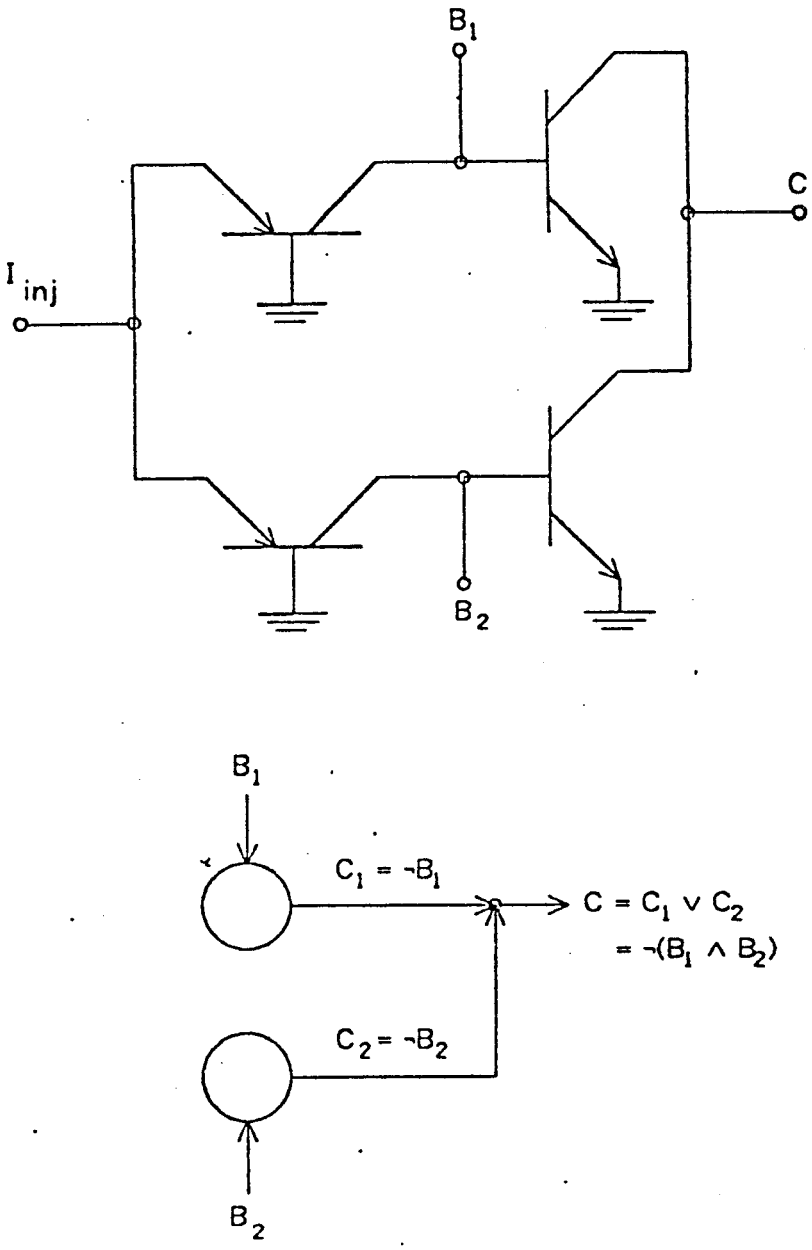

Figure 2-8: I^2L nand gate illustrating logic fanin on a conductor, with corresponding communication graph.

Time. The proper duration for a time unit is derived from the bandwidth limitation of wires, as expressed in assumption L2. A wire in a communication graph has at most unit bandwidth; the model's time scale must be adjusted until this assumption is satisfied.

The effective bandwidth of a conductor on a chip is determined by the signal

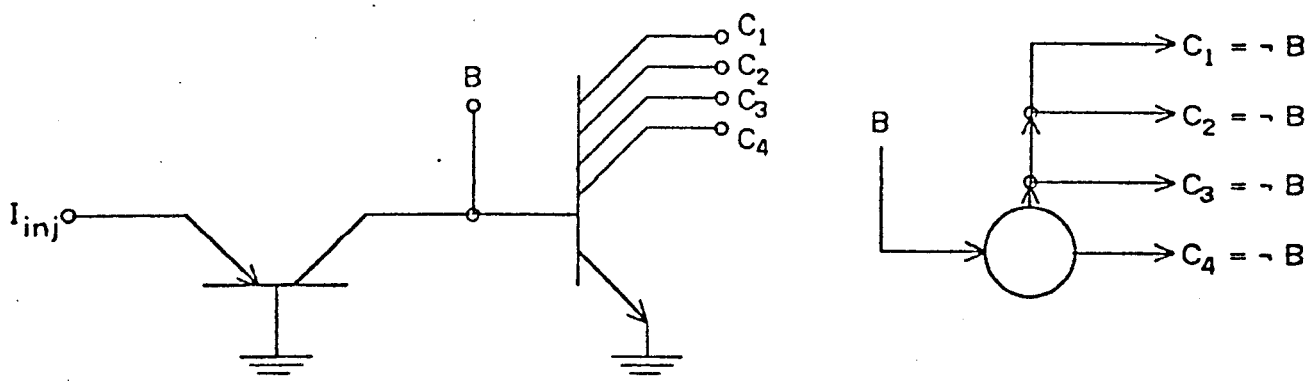

Figure 2-9: I^2L transistor illustrating large fanout at a single device, with corresponding communication graph.

conventions employed by the VLSI designer, and the timing characteristics of the implementing technology. A signal convention is an association of a logical (boolean) value with a voltage range or current flow along a wire. A set of signal conventions thus provides a correspondence between the logical signals on the wires in a communication graph, and the dynamic electrical "states" of conductors on the surface of a VLSI chip. Each signal has a time dimension, in the sense that a changed voltage or current configuration must be maintained for some minimal duration of time before it amounts to a changed logical value.

The bandwidth of a conductor is the average information content of each signal divided by the duration of a signal. If each signal is equally likely to occur from the receiver's point of view, then the information content of each signal is the (base two) logarithm of the number of possible signals. Any other probability distribution for the signals gives a smaller information content and a smaller effective bandwidth. See [Shannon 49], p. 21.

The lower bound model of computation models the state of a wire as a binary variable, according to assumption L5. This leads to some difficulty in modeling circuits whose wires carry more than one bit per signal. For such circuits, the definition of the time unit is modified so that their wires have at most unit bandwidth. The unit of time for the lower bound model is accordingly set equal to the minimal duration of a signal on a wire, divided by the logarithm of the number

of such signals. Scaling the time unit in this way means that a signal carrying k bits of information is modeled indirectly, by k one-bit transmissions.

The predominant two-valued logic is modeled directly. Signals in the model correspond one-to-one with signals on the chip. The unit of time is equal to the clock period, for synchronous logic. The effective time unit for an asynchronous circuit is usually determined by the rate at which data is fed to the circuit, but in any event it cannot be shorter than the delay through one stage of on-chip logic.

TTL-style "Tri-State" logic and other signal conventions with a high impedance state can also be modeled directly. The conductors in such circuits may send one bit of information in both directions simultaneously. Consider the "wire-anded" MOS circuit of Figure 2-10. The outputs C and D go to V_{dd} (logical *1*) iff both A and B are low, placing their transistors in the high impedance state. The wire running between the two transistors must be carrying information in both directions, since C and D depend on both A and B. (If either C or D were unused, unidirectional information flow would suffice to model the circuit.)

Delay. In addition to the bandwidth constraint, time considerations enter the VLSI model in the form of wire delays. (Bandwidth is the amount of information emerging from the end of a wire in unit time, while delay is the amount of time that elapses between the transmission and receipt of a signal. Currently, VLSI wires can carry only one signal at a time, so that the delay of a wire determines its bandwidth. This may not always be the case.) According to assumption L5, a signal created at time t appears at the far end of a wire D at time $t+\delta_D$, where δ_D is the non-negative delay of that wire. There is no explicit assignment of delay to each node, even though one could measure a delay from the time at which a signal appears at its inputs to the time at which a valid signal appears at its outputs. Such a "node delay" is of course an important component of the delay between the appearance of a signal at the input of one node and the appearance of a derived signal at the input of a connected node.

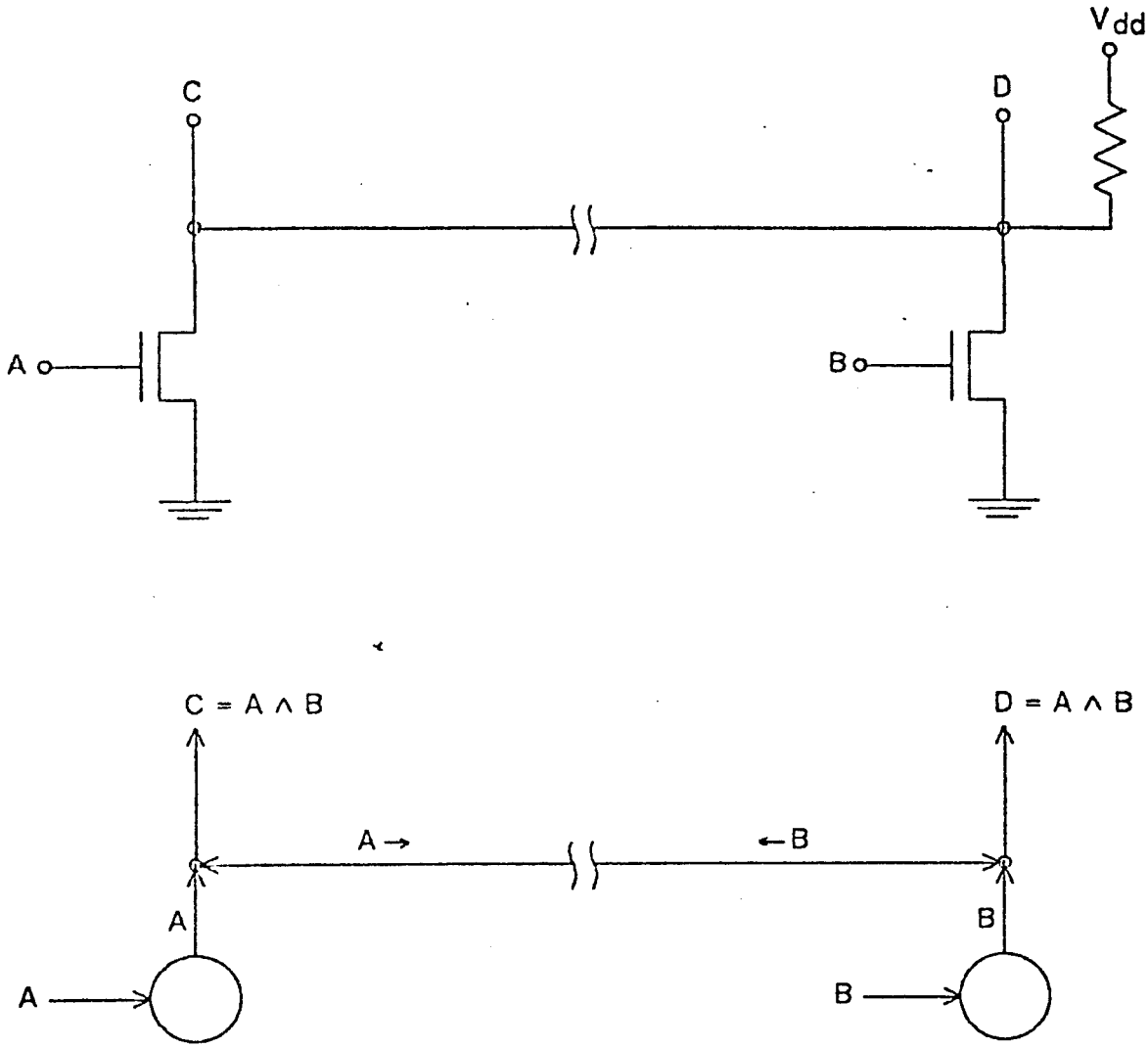

Figure 2-10: MOS "wire-and" circuit, with corresponding communication graph.

The delay of a VLSI conductor is often strongly dependent on its length. This is the reason that a separate delay is associated with each wire.

The delay situation on an actual VLSI chip is really quite complex. Circuit delays are not independent of signal conditions, as suggested by the fixed delays of the VLSI model. Circuits just do not respond to logical *0* and *1* signals at exactly the same rate. The previous state of the circuit is important, and even the states of the surrounding circuits and wires can materially affect delays. However, a single delay value should be quite sufficient to describe the logical design of any chip. (Small variances in delay times will not affect the behavior of a VLSI circuit, for timing variations are cancelled at each synchronization point. The clock period must of course be longer than the longest delay in the circuit.)

It is expected that most chips can be modeled adequately with the following simplistic approach. Every wire emanating from a node that models an active logic element is assigned unit delay. Wires that connect to nodes that model wire junctions have zero delay.

Multiple layers of interconnection. The embedding rules for communication graphs correspond directly to chips with a single layer of interconnection material. Wires are normally formed in this layer. Cross-overs are built with short runs in another conducting layer, for example, polysilicon in MOS technologies.

When modeling chips with a single layer of interconnection, the unit of length is just the minimal spacing between the centers of parallel conductors. The model's grid of unit squares describes this minimal conductor spacing precisely. At most one wire crosses the unit length edge of any square.

A legal communication graph can be drawn for every VLSI chip that employs multiple layers of interconnection by modifying the correspondence between surface area on the chip and unit squares of its graph. If there are k layers of interconnection, each unit of chip area is modeled by k^2 unit squares. Figure 2-11 shows the way in which three units of chip area map into twenty-seven unit squares, if there are three layers of interconnection. The unit squares in each k-by-k image of a unit of chip area are numbered in a matrix notation, but from bottom to top, left to right. Square *(i,i)* corresponds to the *ith* layer on the chip. Transistors are formed in the first layer, and are thus drawn as nodes in squares *(1,1)*. See Figure 2-12. A conducting path running in the first layer of interconnection is modeled as a wire running through squares *(1,1)* -- to get the connectivity right, it is also allowed to run through squares in row or column *1*. In general, wires formed in the *ith* layer of interconnection are drawn in squares *(i,j)* or *(j,i)*. Connections between adjacent layers of interconnection, or "vias," are drawn as wire bends in non-diagonal squares of a k-by-k image. For example, a bend in square $(i,i+1)$ represents a connection between layers i and $i+1$. A branch or fork in a wire on layer j is modeled by a fanout node in square *(j,j)*.

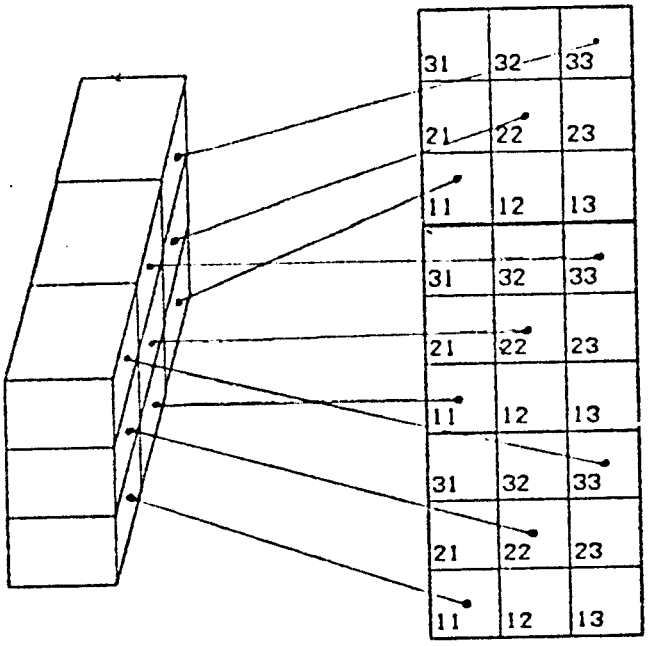

Figure 2-11: Correspondence scheme for a three-layer embedding.

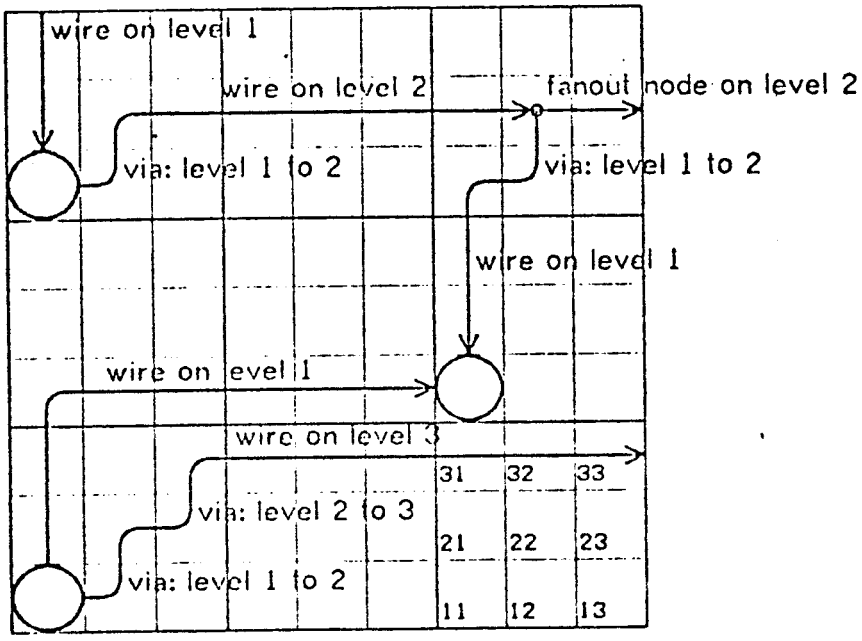

Figure 2-12: A communication graph occupying 9 units of surface area on a three-layer chip.

This correspondence scheme for drawing graphs from chips has an interesting implication for the reverse mapping. If a graph is known to require area A under the normal interpretation of a single level of interconnection, it also must require at least A/k^2 area when k layers of interconnection are available. (Interpret the

minimal embedding of the graph as A/k^2 k-by-k images of unit areas.) The bound is not necessarily tight: it may not be possible to embed the layout in exactly A/k^2 area because of the added restrictions on the placement on nodes and wire bends in k-by-k images. (Also, as a practical matter, "vias" or connections between layers can not be placed arbitrarily close to each other.)

In the limit, the multiple level interpretation gives a "volume" result for graph embeddings. If a graph of area A is embedded in $k=A^{1/4}$ levels, then it must occupy area $A/k^2=A^{1/2}$. Assuming the levels are unit distance apart, the graph must fill an $A^{1/4}$-by-$A^{1/4}$-by-$A^{1/4}$ unit cube, or $A^{3/4}$ units of volume. This is a lower bound result, since the added restrictions on node and wire placement in k-by-k images imply that some planar embeddings in area A do not correspond to legal $A^{1/4}$-level embeddings in volume $A^{3/4}$ (but all embeddings in a $A^{1/4}$-by-$A^{1/4}$-by-$A^{1/4}$ unit cube can be interpreted as an embedding in an $A^{1/2}$-by-$A^{1/2}$ square).

Input registers, source nodes. Any VLSI chip that can solve an N-input problem must have N input registers to store the values of the input data. These input registers correspond to the source nodes of the communication graph for that chip.

Input registers appear as one of two structures on a chip, depending upon how information is stored. A bit of storage can be encoded as a static charge, or lack thereof, on a capacitive element. This approach is common in MOS technologies, in which the gate of a transistor can serve as a storage capacitor. A bank of transistors may thus be an input register. Alternatively, the status of a current flow may define a bit. This dynamic representation is available in any technology. A positive feedback loop, or "cross-coupled logic," can make a circuit bistable. The state of such a circuit encodes one bit; a bank of them encodes an entire input value.

The conductors emanating from an input register correspond to wires coming out of a source node. This correspondence points up one difficulty with the representation of an input register by a single source node. An input register formed

from a bank of storage cells is distributed in space and may have a large number of emanating conductive paths. A source node is localized to a unit square and can have only four outgoing wires. A more elaborate correspondence scheme is needed to handle this discrepancy.

The best model for a k-bit input register is a single source node connected to $k-1$ "auxiliary" nodes. Each node corresponds to one cell of the input register. Presumably, there are conductive paths linking the cells of the input register, so the nodes and wires of the model take up no more area than the input register. The conductive paths emanating from each cell of the input register may be directly modeled as wires connecting to the corresponding node.

This model will overestimate the time taken by the chip, if all the bits of the input register are transmitted immediately. This error is limited, however, to the $k-1$ time units it takes for each of the auxiliary nodes to get its bit from the source node. Such an error is negligible in comparison with the total solution time for the problems treated here. In terms of the methodology presented at the beginning of this chapter, (A,T-K,P,N)-VLSI chips are a subset of (A,T,P,N)-communication graphs, although some (A,T,P,N)-VLSI chips lie outside of this set.

The VLSI model of computation can be extended to handle the case that problem input values are obtained from off-chip connections. A source node is drawn at the point of entry of each input value. On a VLSI chip, such a "point of entry" is a very large contact pad connected to an external wire. More than one input value may come through each contact pad, but there should be no trouble finding room for an equivalent number of source nodes in the image of a contact pad that is many hundreds of unit squares in area. A situation that causes a little more difficulty occurs when information about a single input value is obtained from several different contact pads (for example, the *kth* bit of each input variable might be received on the *kth* contact pad). Since all information about an input variable must be modeled as originating from a single point, extra connections must be included in the communication graph between the images of contact pads. In this way, the

receipt of the *kth* bit of an input variable at a contact pad can be modeled by a signal from an on-chip wire, originating from the source node for that input variable. Fortunately, the data rate of an off-chip connection is comparable to the unit bandwidth of an on-chip wire [Mead 80], so that very few wires will be needed to model these imaginary connections between contact pads. Brent and Kung [Brent 79] model some of the aspects of off-chip communication in a more natural way.

Output registers, sink nodes. The N output values resulting from an on-chip computation must be stored in N output registers on the VLSI chip. The communication graph for such a chip has N *sink nodes*, one for each of these output registers. The output register - sink node correspondence is quite similar to that between input registers and source nodes. Output registers are implemented in the same way as input registers, as banks of storage cells.

When an output register is formed from a bank of k storage cells, it is modeled by a string of $k-1$ auxiliary nodes and one source node. A maximal error of $k-1$ time units can result, as was observed in the modeling of source nodes. This worst case arises if all output cells receive a bit of information in the last operation of the chip. The chip's computation is complete at that time, but the sink node of the model must collect a bit of information from each of the auxiliary nodes. Assuming they are connected linearly, this takes $k-1$ time.

If problem outputs are to be shipped off-chip, sink nodes can be associated with contact pads in a manner analogous to the way in which source nodes were drawn for contact pads. Since a single sink node can handle many output variables, some aspects of the correspondence are simplified. However, connections between contact pads will still have to be introduced if different bits of a single output value are sent off-chip from different contact pads.

2.3 Upper bounds

A communication graph is admissible if it can be implemented as a VLSI chip of the same area and time performance (within constant factors). The upper bound constructions of Chapter 4 are admissible communication graphs and hence correspond to feasible VLSI chips.

The upper bound model of computation consists of assumptions or admissibility rules U1 through U8. Any communication graph satisfying all these rules is admissible. Of course, such a graph must also satisfy assumptions L1 through L8 to be a communication graph in the first place.

The upper bound assumptions were designed to be as simple and general as possible. Their simplicity stems largely from two decisions. All references to electrical parameters (capacitance, current density, etc.) are suppressed, and no attention is paid to constant factors in area or time calculations (the big-O notation is employed throughout). Of course some discussion of electrical and technological parameters is necessary to justify the upper bound assumptions. The constant factors will be largely ignored until Section 4.5, which estimates the actual size of the VLSI circuits proposed in Chapter 4.

The generality of the admissibility rules allows them to apply to all currently feasible VLSI technologies. These include technologies based on metal-oxide and Schottky-barrier field-effect transistors (e.g., the MOS family: CMOS, DMOS, HMOS, NMOS, PMOS, ...) as well as the technologies based on bipolar transistors (such as I^2L).

The first admissibility rule, assumption U1, identifies and defines three types of nodes, *logic nodes*, *driver nodes*, and *receiver nodes*. These are drawn in Figures 2-13 and 2-14. The full text of the rule appears on page 44, in conjunction with its corresponding lower bound assumption, L1.

A logic node must fit in $O(1)$ area, so it can not possibly have more than the $O(1)$ connections allowed by assumption U1. Assumption U5, below, limits the state of a

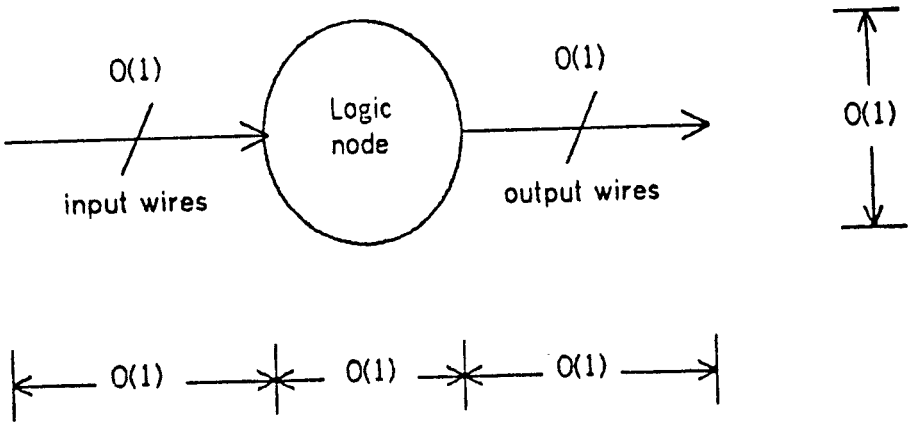

Figure 2-13: Fanin, fanout and dimensions of a logic node.

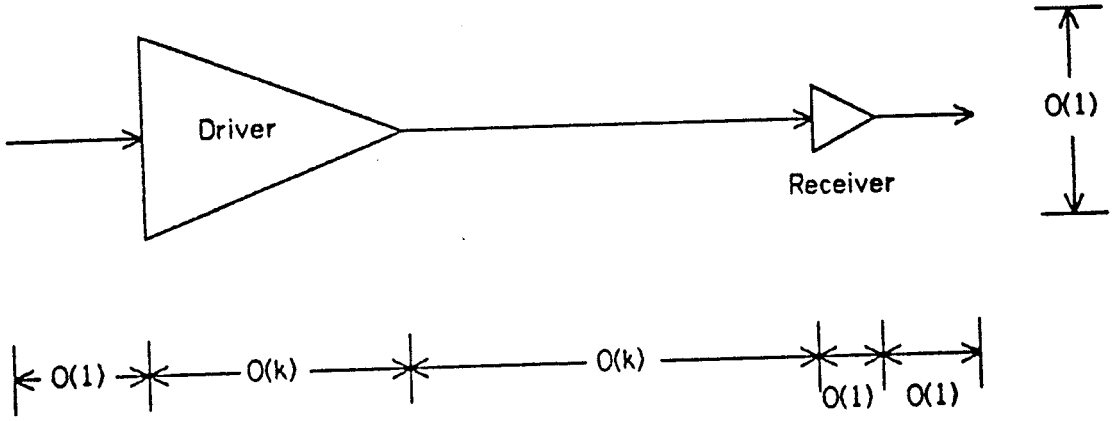

Figure 2-14: Dimensions of driver and receiver nodes.

logic node to *O(1)* bits. Thus *O(1)* transistors suffice to implement the transmission function of any logic node. The transistors can all be minimum-sized, for they do not drive long wires. (The area of a transistor must be proportional to the length of the wire it drives. All wires have a parasitic capacitance that grows linearly with the length of the wire [Mohsen 79].)

When a logic node is actually implemented in VLSI, it will gate its outputs with a locally available clock pulse. The logic node will also need power and ground connections. As mentioned previously, this thesis assumes that some solution will be found for this distribution problem. However, there is no assumption that the phase of the clock signal will be constant over the entire surface of the VLSI circuit, for that may be impossible to arrange. Other methods are necessary to synchronize distant circuits. Long-range synchronization is one of the functions of driver and receiver nodes.

A *self-timed region* or system is defined as a set of logic elements that maintains synchronization internally, but communicates asynchronously with other self-timed regions [Seitz 79]. For the purposes of this thesis, a self-timed region is a set of logic nodes that receive clock pulses with nearly identical phase. Thus all nodes within a self-timed region are in synchronization with each other.

Assumption U1 states that any square region of at most $O(\log^2 N)$ area can qualify as a self-timed region. This assumption is dictated by practical considerations. Since a word of data has $O(\log N)$ bits, a self-timed region is just large enough to perform a significant computation on a few words. The use of smaller self-timed regions would complicate the constructions of Chapter 4. Also, it is unlikely that any circuit would use smaller regions due to the cost of synchronizing signals that cross region boundaries. Larger regions are of course a possibility, but they are not needed for the circuit constructions of this thesis.

The output wire of a *driver* node is the only wire that can cross the boundary of a self-timed region. The *receiver* node attached to this wire must be able to synchronize itself with the driver. Otherwise it might sample the state of its input at a time when it is being changed by the driver. Actual implementations of driver and receiver nodes may employ any one of a number of synchronizing techniques. For example, the driver might send a clock bit between each data bit, and the receiver might sample its input at several times the data transmission rate. Note that this technique will not reduce the bandwidth of a wire by more than a constant factor. Thus even long wires can carry one bit of information in $O(1)$ units of time.

Driver nodes are also distinguished from all other nodes by their ability to drive long wires. However, the size of the driver must increase linearly with the length of its wire. This is a consequence of the capacitive nature of the load presented by a wire in VLSI technologies. A wire of length k has $O(k)$ capacitance, so that $O(k)$ units of drive current are needed to change its state (voltage level) in unit time. This

amount of current can be obtained only from a transistor of $O(k)$ area.[1]

The foregoing discussion is summarized by the following statements of assumptions L1 and U1.

Assumption L1: Area. A unit square can contain one node or one wire cross-over. One wire may cross each edge of a unit square, so that nodes have a maximum of four wires.

Assumption U1: Area. The area of a node is determined by its functionality.

a. A logic node is a node with at most $O(1)$ input wires, $O(1)$ output wires and $O(1)$ area. Each of its wires is $O(1)$ units long, that is, each wire runs through at most a constant number of unit squares. Each logic node belongs to a self-timed region. All wires connecting to a logic node must lead to or from other nodes in its self-timed region. Every self-timed region must be small enough to fit within a square of $O(\log^2 N)$ area.

b. A driver node and a receiver node are associated with each wire that is more than $O(1)$ units long or crosses the boundary of a self-timed region. A wire of length k requires a driver that occupies an $O(1)$ by O(k) unit area. Its receiver node takes up only $O(1)$ units of area. The driver's input wire and the receiver's output wire are $O(1)$ units long.

Arguments can be made for the use of larger delay functions. Since a signal can not travel faster than the speed of light, a length k wire should have delay $O(k)$. Also, since all wires have some resistance in addition to their capacitance, the speed of signal propagation is limited by the diffusion equation: a length k wire has delay $O(k^2)$ [Seitz 79]. The assumption of logarithmic delay is chosen here, as it seems to give the least misleading results. The constant factor in the $O(k^2)$ delay rule would be quite small, even for the largest constructions proposed in this thesis. The timing of these VLSI circuits will probably not be dominated by wire delays.

---

[1] Unfortunately, this strategy for obtaining unit bandwidth on long wires is not quite adequate. A wire can only carry a limited amount of current without damage, so that drivers can not be scaled-up indefinitely. One way of avoiding this difficulty would be to match the impedances of drivers, wires, and receivers. The resulting transmission lines would have unit bandwidth, yet the drivers would no longer have to charge or discharge the entire wire in one time unit. (This discussion is hypothetical, since no current VLSI technology has on-chip transmission lines.)

Assumption U2 ensures that the upper bound constructions lie within a compact region of the plane. The area charge for a star-shaped circuit thus includes the unusable gaps between the points of the star.

> **Assumption L2**: Total area. The total area of a communication graph is equal to the number of unit squares occupied by wires or nodes.
>
> **Assumption U2**: Total area. The total area of an admissible communication graph is the number of unit squares in the smallest bounding rectangle.

Assumption U3 sidesteps the thorny issues of the actual information capacity of a bidirectional (wire-anded) wire.

> **Assumption L3**: Units of time. A wire has at most unit bandwidth in each direction.
>
> **Assumption U3**: Units of time. A wire has at most unit bandwidth in one direction only.

Assumption U4 restricts the problem domain of the upper bound constructions. Restricting the number of inputs N to be a power of two permits the use of the fast Fourier transform algorithm. Restricting the word size, $d\log Me$, to $O(\log N)$ merely simplifies the form of the upper bounds, which would otherwise have a dependence on M as well as N.

> **Assumption L4**: Problem definition. Each of N input variables takes on one of M different values, for a total of M^N equally likely problem instances. In the sorting problem, $M = N^{1+\varepsilon}$ for some fixed positive ε. In the Fourier transformation problem, $M > N$.
>
> **Assumption U4**: Problem definition. $N = 2^N$ and $\log M = O(\log N)$.

Assumption U5 defines the delay characteristics of admissible communication graphs. Logic nodes have at most $O(1)$ delay. Greater delays are inconceivable, since a logic node has only $O(1)$ transistors and thus cannot "count" more than a constant number of clock pulses.

The delay of a driver-wire-receiver circuit is proportional to the logarithm of the length of the wire. This assumption is consistent with the use of $O(k)$-area drivers

for length-k wires. The input to a driver nodes comes from a logic node composed of minimum-sized transistors. This signal must be amplified by $O(\log k)$ stages before it can be sent down the long wire. Each stage contributes $O(1)$ units of delay and is $O(1)$ times the area of the previous stage. If, additionally, each stage is clocked so that the amplification chain becomes a pipeline with $O(\log k)$ bits of capacity, the driver-wire-receiver circuit attains unit bandwidth and $O(\log k)$ delay.

Arguments can be made for the use of larger delay functions. Since a signal cannot travel faster than the speed of light, a length k wire should have delay $O(k)$. Also, since all wires have some resistance in addition to their capacitance, the speed of signal propagation is limited by the diffusion equation: a length k wire has delay $O(k^2)$ [Seitz 79]. However, the constant factor associated with an $O(k)$ or $O(k^2)$ rule is so small that this thesis' assumption of $O(\log k)$ delay is the most appropriate.

Assumption L5: Transmission functions. Node states and wire signals are completely and consistently described by the transmission function associated with each node. A node with state vector S, input wires $(A, \ldots, D)$, and output wires $(E, \ldots, G)$ computes a function of the form

$$[S(t+1), E(t+\delta_E), \ldots, G(t+\delta_G)] = F[S(t), A(t), \ldots, D(t)],$$

where δ_E and δ_G are the non-negative delays of wires E and G.

Assumption U5: Transmission functions. The transmission function of a node is constrained by its functionality.

a. A logic node has at most $O(1)$ bits of state and $O(1)$ units of delay on each of its output wries.

b. The total delay through a driver-wire-receiver circuit is $O(\log k)$ if the wire is k units in length. The driver and receiver nodes associated with this wire implement the identity function, so that the receiver output $R(t)$ is a delayed version of the driver input $D(t)$. The combined transmission function of the driver and the receiver is

$$R(t + \delta_w) = D(t), \qquad (2.3)$$

where $\delta_w = O(\log k)$.

Assumption U6 relaxes an idealization of the lower bound model, that there can

be "point sources" of $d\log Me$ bits of information. Since logic nodes have only $O(1)$ bits of state, $O(\log M)$ of them are needed to store a problem input value.

Assumption L6: Source nodes. The initial state of a source node may be any function of the value of its input variable. Each input variable affects only the initial state of its source node.

Assumption U6: Source nodes, input registers. A source node is the middle member of a string of $\lceil \log M \rceil$ logic nodes called an *input register.* The initial state of the kth node of an input register is equal to the kth bit of the binary expansion of the value of its input variable, $1 \leq k \leq \lceil \log M \rceil$. (Note that this assumption violates assumption L6, although its only effect is to allow an admissible communication graph to be initially in a state that a legal communication graph could only reach after $\lceil \log M \rceil / 2$ units of time.)

Assumption U7 is analogous to Assumption U6. It relaxes the lower bound assumption of "point sinks" of information.

Assumption L7: Sink nodes. There is a fixed assertion for each sink node, relating its state to the correct values of its output variables (as a function of the values of the input variables). A computation is complete at time T if all assertions are satisfied at all times $t \geq T$.

Assumption U7: Sink nodes, output registers. A sink node is the middle member of a string of $\lceil \log M \rceil$ logic nodes called an *output register.* The computation is complete when the kth node of every output register contains the correct value of the kth bit of its output variable, $1 \leq k \leq \lceil \log M \rceil$, as defined by the output assertion of its sink node. (This assumption violates L7, since an admissible communication graph is allowed to anticipate a legal completion by $\lceil \log M \rceil / 2$ time units.)

There is no assumption U8. Problem solution time is defined the same way for upper bounds as it is for lower bounds.

Assumption L8: Problem solutions. A communication graph is said to solve a problem in worst-case time T if it takes no longer than T units of time to complete its computation of any problem instance. A communication graph is said to solve a problem in average time T if its average completion time, over all problem instances, is T.

2.4 Relation to the model of Mead and Rem

Mead and Rem [Mead 79] have optimized the area-time product for memory chips, using a slightly different model of VLSI circuitry. Their unit of length is the same as that of this thesis, the minimum distance between two wires. Their timing rules can be summarized in the following statement. A 'bus wire' of length αb that has α driving transistors of area b and one receiver of area αb operates in α time units. For example, a unit-sized transistor can drive another unit-sized transistor at the end of a unit-length wire in just one time unit. In terms of the above rule, $\alpha=b=1$. Alternatively, a 'bus driver tree' with a branching factor of α has a delay of α for each level that a signal ascends from a leaf-node driver $(b=1)$.

Mead and Rem could have reached similar results for their area-time analysis using the VLSI model of computation. Their area estimates are always obtained by counting wires, never by the size of the drivers. As for the timing rule, it is merely a worst-case result for the communication graph of Figure 2-15. A signal from the right-most of α leaf nodes takes α time units to get to its parent.

Despite the fact that the VLSI model of computation draws heavily on the work of Mead and Rem, their model is quite different in character. They build a single technology-dependent model instead of building two separate but general models for upper and lower bounds. In this way, they are able to predict the size of their constructions quite precisely, whereas the upper bounds developed here are accurate only to within a constant factor. On the debit side, their preoccupation with the MOS transistor as a basic building block complicates their model and limits its range of application.

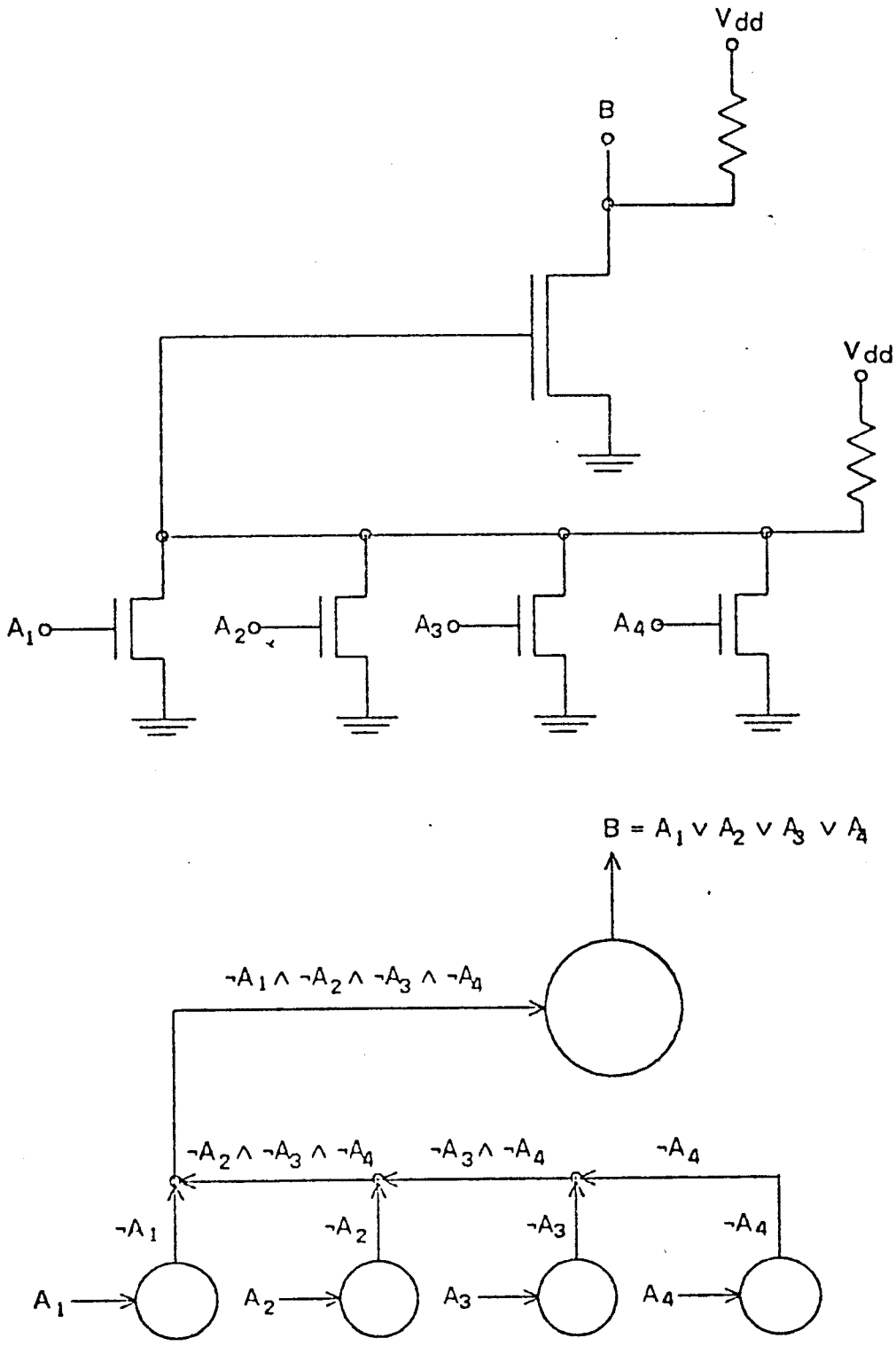

Figure 2-15: A bus wire drawn as a communication graph ($\alpha = 4, b = 1$).

Chapter 3
Lower bounds

Lower bounds are proved in the VLSI model of computation by considering all possible communication graphs. A key property of each graph is identified, its *minimum bisection width.* Intuitively, the wider a graph, the more bandwidth it has across its midsection. Thus wide graphs are generally faster but larger than narrow graphs. More precisely, the area and time performance of a communication graph can be bounded from a knowledge of its minimum bisection width. The area of a graph is at least proportional to the square of its width, $A \geq \omega^2/4$, as proved in Theorem 2. The time taken by a communication graph to solve an N-point DFT or sorting problem is at least inversely proportional to its width: $T = \Omega(N \log N)/\omega$, by Theorems 10 and 15. The two theorems combine immediately to form the lower bound result $AT^2 = \Omega(N^2 \log^2 N)$. By assumption L6 of Section 2.1, there are N source nodes in a graph solving an N-input problem, so that $A \geq N$. On the basis of this area bound and the area and time results quoted above, Theorem 12 proves that the minimum value for a performance metric of the form AT^{2x} occurs when $\omega = \theta(N^{1/2})$, leading to the general lower bound $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$ for $0 \leq x \leq 1$.

This chapter is divided into three sections. The first defines the minimum bisection width of a graph. The next proves a lower bound on graph area in terms of its width. Section 3.3 derives bounds on graph speed, given its width, for the problems of Fourier transformation and sorting. Combined area-time bounds follow as simple corollaries to these results.

3.1 Minimum bisection width

The minimum bisection width of a graph is, informally, the number of edge cuts needed to slice it in half. In other words, it is the smallest number of edges whose removal disconnects one half of the vertices from the other. For example, the minimum bisection width of a linear graph or complete binary tree of N vertices is *1*, while the "mesh" of $N = n^2$ vertices has width $n + (n \bmod 2)$. See Figure 3-1.

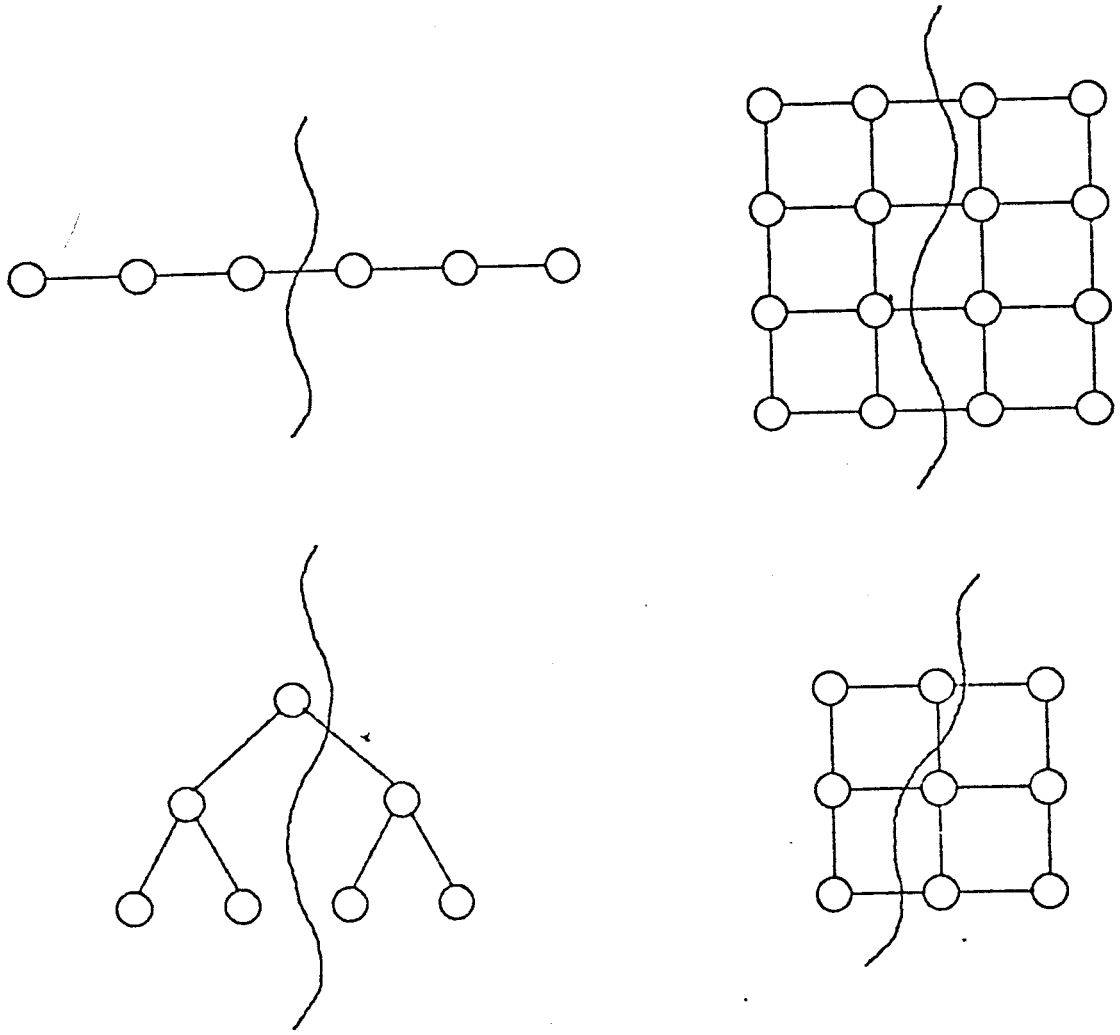

Figure 3-1: Sample minimum bisections.

A slightly more general concept of bisection is needed for the lower bound proofs of Section 3.3. For reasons that will become apparent later, a communication graph is only "bisected" if half of the source nodes lie on either side of the bisection. This

idea is formalized in the following definitions, written in standard graph-theoretic terminology. (A communication graph corresponds to an undirected graph of maximum degree *4*, when nodes are replaced by vertices and wires by undirected edges. Vertices corresponding to source nodes are members of the set S referred to below.)

Let G be an undirected graph with vertices V and edges E. Let $S \subseteq V$ be a subset of the vertices, and $E_S \subseteq$ E be a subset of the edges in G. Then E_S is said to "bisect S in G" if the removal of E_S induces some partition of V into two sets of vertices V_1 and V_2, each containing approximately half the vertices in S, such that

1. $V_1 \cup V_2 = V, \quad S_1 \cup S_2 = S,$
2. $S_1 \subseteq V_1, \quad S_2 \subseteq V_2,$
3. $|S_1| \leq |S_2| \leq |S_1| + 1$, and
4. Every path from a vertex in V_1 to a vertex in V_2 contains an edge in E_S. (A path exists from vertex x to vertex y if $x = y$, or if there is some vertex z such that (x,z) is an edge and there is a path from z to y.)

The minimum bisection width of S in G is defined as the number of edges in the smallest cutset E_S bisecting S in G. Formally,

$$MBW(S,G) = min\{|E_S| \text{ s.t. } E_S \text{ bisects } S \text{ in } G\}. \qquad (3.1)$$

It is quite difficult in general to compute the minimum bisection width of a graph. In fact the problem is NP-complete, as shown by Garey's proof [Garey 74] of the completeness of "minimum cut into equal-sized subsets." Fortunately, it is enough for most purposes to know that every graph has a set of edges that realizes its minimum bisection width.

3.2 Area

The area occupied by a communication graph is defined by the assumptions of Section 2.1. In brief, area is measured by the number of unit squares filled by wires or nodes. Theorem 2 proves that any communication graph with minimum bisection width ω must occupy at least $\omega^2/4$ unit squares. Before proceeding with the proof of that theorem, we prove the following result to develop the reader's intuition.

Theorem 1: If a communication graph fits within a rectangle of area $(\omega-1)^2$, then it can be bisected by cutting at most ω wires.

Proof. Rotate the communication graph by ninety degrees, if necessary, so that the height of the bounding rectangle is at most $\omega-1$. Next, try to position a vertical line within the rectangle so that half of the source nodes are on either side. Such a position may not exist, but it is easy to see that a vertical zig-zag line with a unit-length horizontal "step" can bisect any graph, as illustrated in Figure 3-2. The total length of the bisecting zig-zag inside of the rectangle is at most ω. By assumption L2, at most one wire can cross any unit-length horizontal or vertical line segment, so that the bisecting zig-zag cuts at most ω wires. □

Note that the contrapositive of Theorem 1 is "if the minimum bisection width of a communication graph is $\omega+1$, then its minimum enclosing rectangle has an area of greater than $(\omega-1)^2$ unit squares." This is the area of a communication graph for upper bound purposes, as defined by assumption U2 of Section 2.3. The following result puts a *lower* bound on the area of a communication graph, in accordance with assumption L2 of Section 2.1.

Theorem 2: If the minimum bisection width of the source nodes in a communication graph is ω, then the wires and nodes of the graph must occupy at least $\omega^2/4$ unit squares.

Proof. Place a Cartesian coordinate system on the grid of unit squares in such a way that the corners of all squares have integer coordinates. Nodes, being at the center of squares, lie wholly to the left or right of any vertical line $\{x=i\}$, when i is an integer.

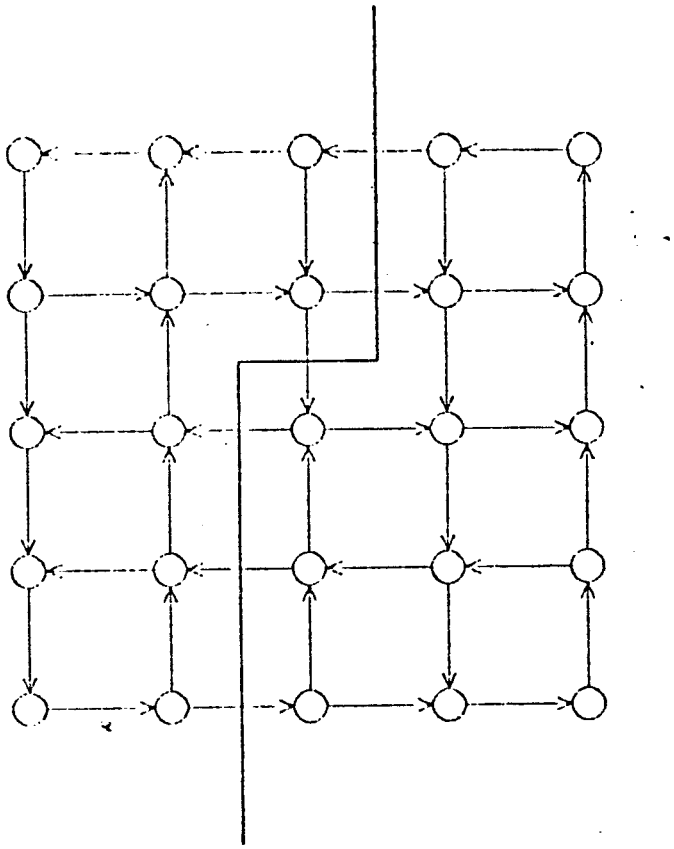

Figure 3-2: A zig-zag bisecting a communication graph.

Overview. The idea behind the proof is the construction of ω^2 bisections of the graph, each bisection defined by a "zig-zag" line similar to the one used in the proof of Theorem 1. For example, Figure 3-3 shows an embedding of the shuffle-exchange graph of sixteen nodes, bisected in four different ways by the heavy "zig-zags" numbered *1* through *4*. (The subscripted *i* and *j* values define the positions of these zig-zags according to the notation developed later in the proof.) In general, each zig-zag cuts the plane into two pieces, each of which has about half of the source nodes. Each zig-zag must cut at least ω wires, by definition of the minimum bisection width. The outermost vertical sections of the zig-zags are disjoint, so that $\theta(\omega)$ occupied squares may be associated with the $\theta(\omega)$ wires cut by these vertical sections, for a grand total of $\theta(\omega^2)$ occupied squares. Figure 3-4 illustrates the way in which squares can be associated with most places that a wire may cross a zig-zag.

The First zig-zag. The first zig-zag to be constructed has a horizontal segment one

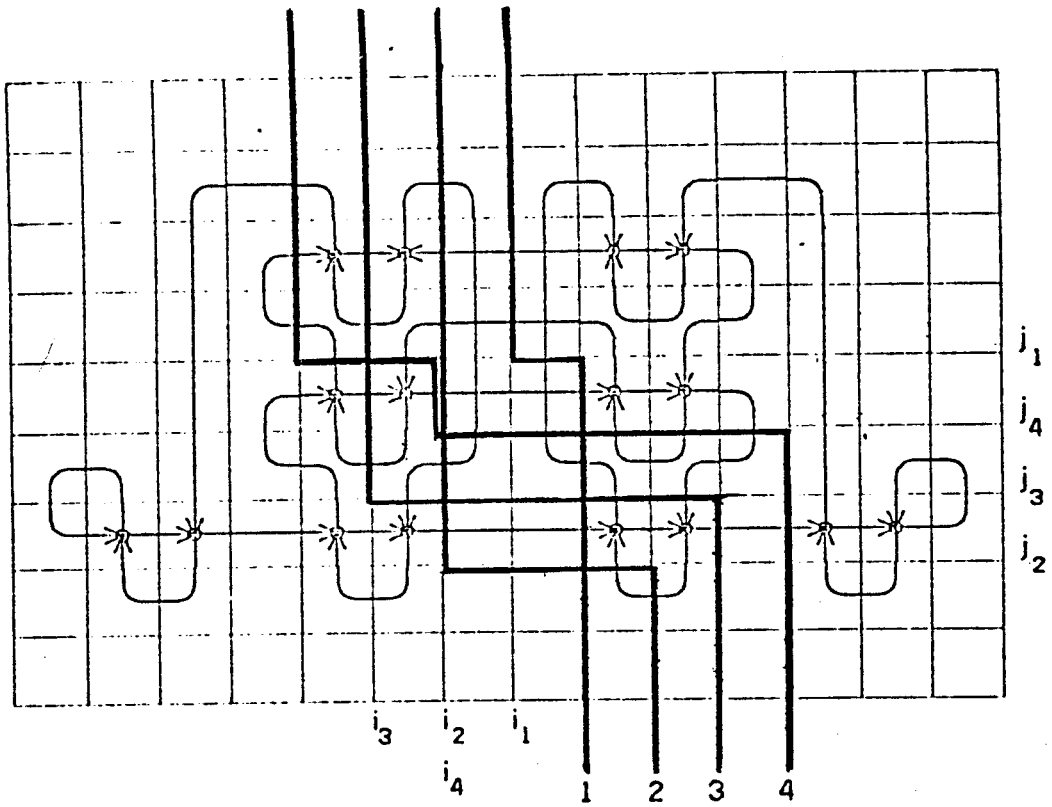

Figure 3-3: Four zig-zags.

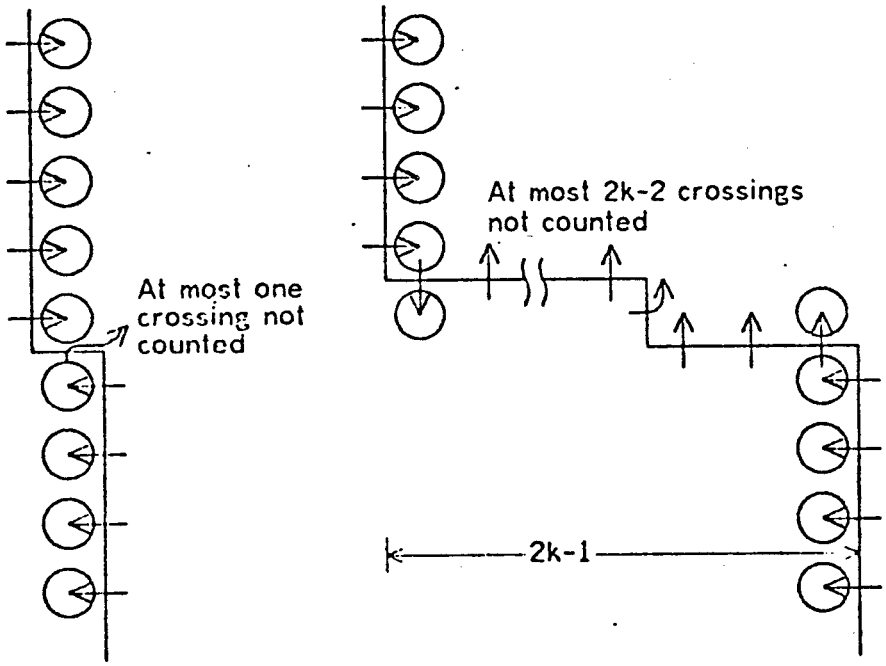

Figure 3-4: First and *kth* zig-zags, showing occupied squares for most wire crossings.

unit in length. The following paragraph contains a formal description of this zig-zag, and a proof that it can be placed to bisect any graph.

Let $L(i)$ be the set of source nodes to the left of the line $\{x=i\}$. By monotonicity, there is some value i_1 that nearly bisects the N source nodes, or more precisely, that satisfies $|L(i_1)| \leq N/2$ and $|L(i_1+1)| \geq N/2$. A related exact bisection is achieved by some zig-zag of the form shown in Figure 3-5. As j_1 increases, the number of source nodes to the left of the zig-zag increases monotonically.

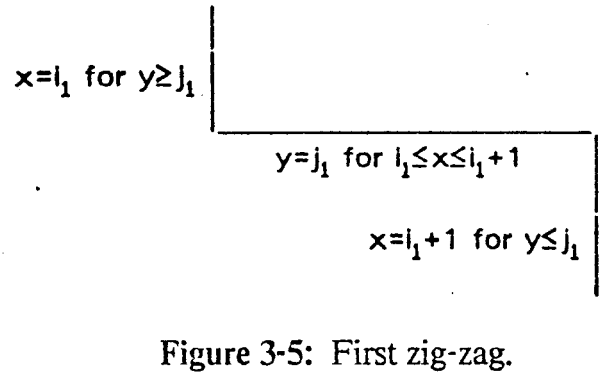

Figure 3-5: First zig-zag.

By the definition of the minimum bisection width, any bisection of the graph cuts at least ω wires. Any wire that crosses the zig-zag must run through one of the unit squares lying between the lines $\{x=i_1\}$ and $\{x=i_1+1\}$. At least $\omega-1$ of these unit squares must be occupied by wires or nodes, since all wires intersected by vertical portions of the zig-zag run into disjoint squares, and at most one wire can intersect the horizontal segment of the zig-zag.

Later zig-zags. Other zig-zags can be drawn to argue the existence of occupied squares outside the column defined by $\{i_1 \leq x \leq i_1+1\}$. By monotonicity, a j_2 can be found for which the zig-zag shown in Figure 3-6 nearly bisects the graph. The set $L(j_2)$ of vertices to the left of the zig-zag satisfies $|L(j_2)| \leq N/2$ and $|L(j_2+1)| \geq N/2$. An exact bisection can be obtained by the introduction of yet another bend (a "step") in the zig-zag. An i_2 can be found in the range $i_1-1 \leq i_2 \leq i_1+2$ such that the zig-zag of Figure 3-7 defines a bisection.

All wires that cross this zig-zag must run through one of the unit squares in the columns $\{i_1-1 \leq x \leq i_1\}$ or $\{i_1+1 \leq x \leq i_1+2\}$. In fact, $\omega-2$ squares of these

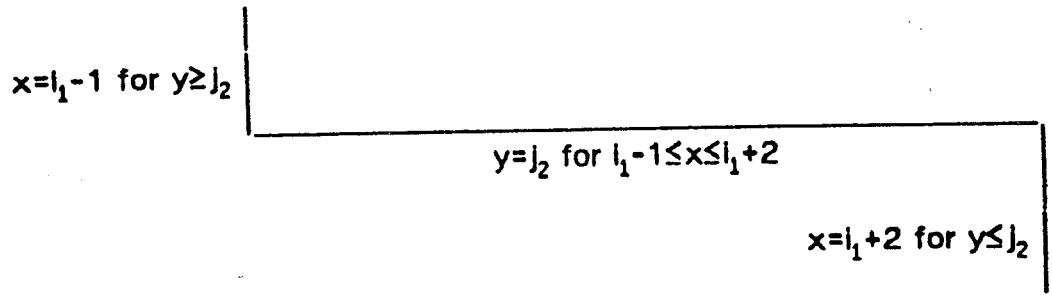

Figure 3-6: First attempt at second zig-zag.

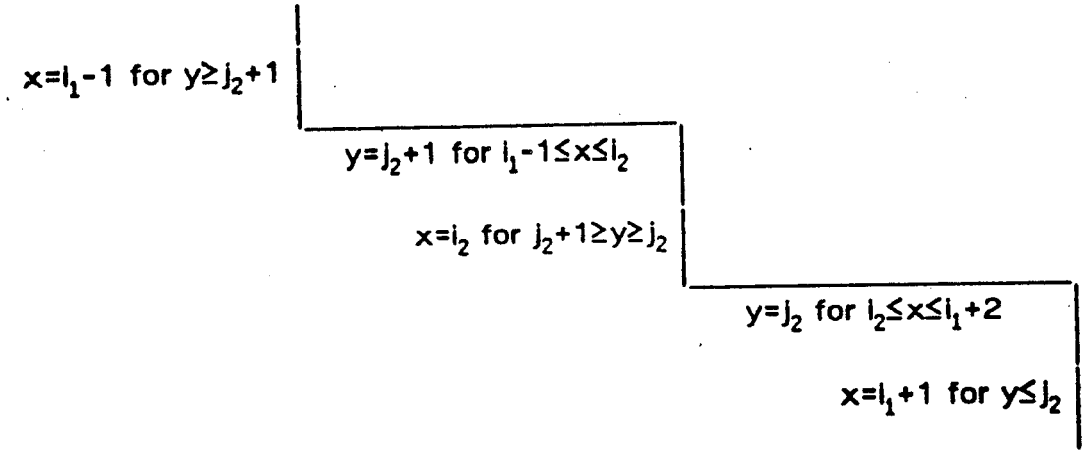

Figure 3-7: Second zig-zag, with step.

columns must be occupied, since at most two wires may pass through the central portion of the zig-zag. See Figure 3-4.

In all, a total of $\lfloor \omega/2 \rfloor$ zig-zags can be drawn to bisect the graph. The *kth* zig-zag has the form shown in Figure 3-8. The long vertical segments of the *kth* zig-zag are $2k-1$ units apart, so that at most $2k-3$ wires can pass through the central $2k-3$ units of its horizontal steps without being counted by the scheme of Figure 3-4. Additionally, one wire can cross its unit-length vertical step. Out of a total of at least ω wires crossing the *kth* zig-zag, at least $\omega-2k+2$ of them must occupy squares in the columns $\{i_1-k+1 \le x \le i_1-k+2\}$ and $\{i_1+k-1 \le x \le i_1+k\}$.

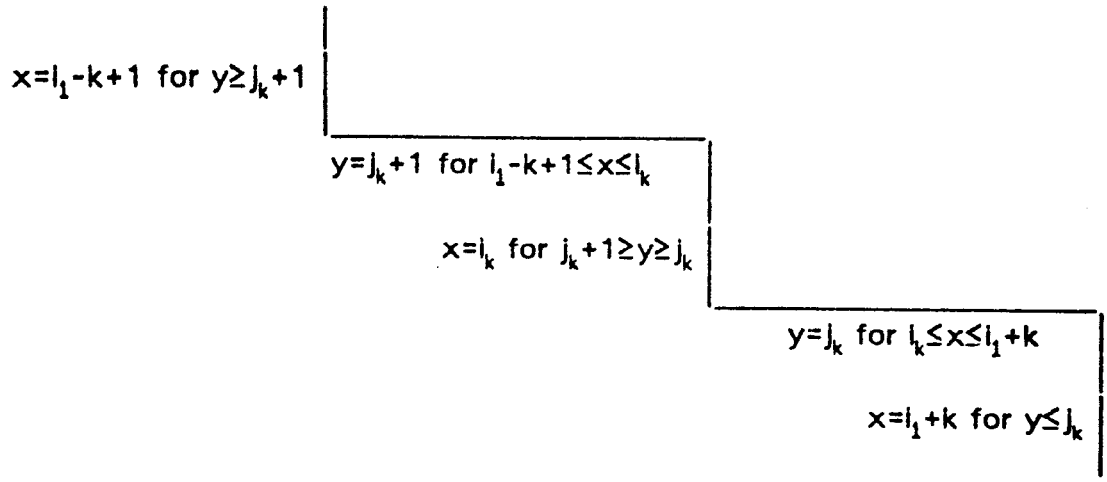

Figure 3-8: *Kth* zig-zag, for $k \geq 2$.

Summary. The sum of the occupied squares counted by these zig-zags puts a lower bound on total graph area of

$$\omega - 1 + \sum_{1<k\leq\lfloor\omega/2\rfloor}(\omega - 2k + 2) \geq \omega^2/4, \qquad (3.2)$$

for $\omega \geq 2$. The theorem also holds for the trivial case $\omega = 1$. □

The zig-zag constructions of Theorem 2 can also be used to prove a more general graph embedding result. The most natural statement of the following theorem defines a slightly different set of embedding rules from any considered elsewhere in the thesis. Here, k edges are allowed to pass over any point in the plane, so that $k=1$ corresponds to a strictly planar embedding. When $k=2$, edges are allowed to run on top of each other for any distance, while the wires of Section 2.1 may only pass over each other at crossovers. Finally, vertices may have degree $4k$ (whereas nodes have maximum degree 4), because k edges can cross each edge of the unit square containing a vertex.

Theorem 3: A graph $G = (V,E)$ with maximum node degree $4k$ and minimum bisection width $MBW(V,G) = \omega$ cannot be embedded in less than $\omega^2/(4k^2)$ units of area, if vertices have unit area, edges have unit width, and no more than k edges pass over any point in the plane.

Proof. The idea behind the proof is to construct a number of bisecting zig-zags,

as in the proof of Theorem 2. In this case, k edges can cross each unit length of a zig-zag. The *jth* of $\lfloor \omega /2k \rfloor$ zig-zags will demonstrate the existence of $\omega/k - 2(j-1)$ occupied squares.

The first zig-zag cuts ω edges, of which at most k can cross the horizontal portion. It thus demonstrates the existence of $\lceil (\omega - k)/k \rceil \geq \omega/k - 1$ occupied squares of the central column of the embedded graph. The second zig-zag also cuts ω edges, but $2k$ of these may cross the central portion without necessarily going through any of the squares flanking the central column. The second zig-zag thus accounts for $\lceil (\omega - 2k)/k \rceil \geq \omega/k - 2$ occupied squares. In general, the *jth* zig-zag accounts for $\lceil (\omega - 2(j-1)k)/k \rceil \geq \omega/k - 2(j-1)$ squares. The sum of all these contributions for $1 \leq j \leq \lfloor \omega/2k \rfloor$ is at least $\omega^2/(4k^2)$, hence the theorem. □

3.3 Time bounds

The previous section related the minimum bisection width of a communication graph to its area. This section develops a complementary theorem linking the width of a graph to the time it takes to compute its function. The two results together give lower bounds on area-time complexity.

By definition, a communication graph with minimum bisection width ω can be partitioned into two subgraphs, each containing half the source nodes, upon the removal of only ω wires. This partition corresponds to a bisection of the problem being solved, $\tilde{y} = f(\tilde{x})$. If k of the sink nodes and $\lceil N/2 \rceil$ of the source nodes are on "side R" of a bisected graph, then the other $N-k$ sinks and $\lceil N/2 \rceil$ sources are on the other side, "S." Side R must compute values for the k variables in $\tilde{y}_R$ corresponding to the k sink nodes included in its half of the bisected graph. The values of the input variables $\tilde{x}_R$ are available to side R; as they are stored in its $\lceil N/2 \rceil$ source nodes, but it has no initial information about the input values $\tilde{x}_S$ on the far side of the bisection. In general, side R will need some information about $\tilde{x}_S$ to compute the correct values for its outputs $\tilde{y}_R$; however, the amount of information needed may be affected by the known inputs $\tilde{x}_R$. (For example, if side R is to compute the

$\lceil N/2 \rceil$ smallest outputs in a sorting problem, it needs no information about the other $\lfloor N/2 \rfloor$ inputs if all of its own inputs $\tilde{x}_R$ are zero.) These considerations motivate the following definition of a subfunction $\tilde{y}_R = F_R(x_R, *)$ associated with each bisection R and each assignment of values to $\tilde{x}_R$:

$$\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S) \Big|_{\tilde{x}_R}, \tag{3.3}$$

where

$$|\tilde{y}_R| = k, \ |\tilde{x}_R| = \lceil N/2 \rceil, \text{ and } |\tilde{x}_S| = \lfloor N/2 \rfloor. \tag{3.4}$$

A lower bound on information flow between the two halves of the communication graph can be obtained by arguments on the structure of the $F_R(\tilde{x}_R, *)$ arising from a bisected function F.

One possibility is that no information flow is necessary. This case arises whenever $F_R(\tilde{x}_R, \tilde{x}_S)$ evaluates to a constant, independent of the value of $\tilde{x}_S$. For example, the problem of computing a simple transformation $\tilde{y}=a\tilde{x}$ is perfectly partitionable with no necessary information flow. Any bisection that places corresponding elements of $\tilde{x}$ and $\tilde{y}$ on the same side allows that side to compute its result without any information about the elements of $\tilde{x}$ that are on the other side.

The other extreme case is that the computation of $\tilde{y}_R$ requires complete information about $\tilde{x}_S$. In other words, all the functions $\tilde{y}_R = F_R(\tilde{x}_R, *)$ are injective. Each of the $|\{\tilde{x}_S\}|$ possible values of $\tilde{x}_S$ leads to a different value of $\tilde{y}_R$, for any fixed $\tilde{x}_R$.

> **Lemma 4:** If the minimum bisection R of a communication graph of width ω induces an injective function $\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S)$ for each $\tilde{x}_R$, and if all $|\{\tilde{x}_S\}|$ values of $\tilde{x}_S$ are equally likely, then the average time to compute F is at least $(\log |\{\tilde{x}_S\}|)/\omega$.

Proof. Any communication between the source nodes for $\tilde{x}_S$ and the sink nodes for $\tilde{y}_R$ must pass over the ω wires defining the minimum bisection. Total bandwidth is thus limited to ω bits per time unit. Over this channel must pass information to disambiguate $|\{\tilde{x}_S\}|$ equally likely possibilities, so that the proper value of $\tilde{y}_R$ may be determined.

As indicated in Section 1.1, the entropy H of a probability distribution is $\Sigma -p_i \log p_i$. In this case, $p_i = 1/|\{\tilde{x}_S\}|$, so that $H = \log |\{\tilde{x}_S\}|$. By Theorem 9 (page 28) of [Shannon 49], a channel with bandwidth ω cannot transmit at an average rate exceeding ω/H, no matter what coding scheme is used. The average time to determine a $\tilde{y}_R$ value is thus at least as long as the inverse of this average rate, or $(\log |\{\tilde{x}_S\}|)/\omega$ time units. □

This lemma indicates a way to show that a function F takes much time to compute. One could demonstrate that, no matter how the inputs $\tilde{x}$ are bisected, the resulting $F_R(\tilde{x}_R, *)$ are all injective.

In fact, much weaker conditions will suffice. To require injectivity is, from an information-theoretic standpoint, to require a trivial mapping between input and output probabilities. In an injective mapping, the probability of a specific output is equal to the probability of its corresponding input.

Non-injective mappings define more complex transformations on probability spaces. Assuming that each input value is equally likely, the probability of an output value is proportional to the number of inputs that produce that output. In this way, a conditional probability $p(\tilde{y}_R | \tilde{x}_R)$ can be evaluated for each value of $\tilde{y}_R$, given a value for $\tilde{x}_R$, by counting the number of inputs $\tilde{x}_S$ for which $\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S)$.

Side R of a communication graph must receive enough information from side S to determine its outputs, $\tilde{y}_R$. At best, side S could send just

$$\sum_{\tilde{y}_R} -p(\tilde{y}_R | \tilde{x}_R) \log p(\tilde{y}_R | \tilde{x}_R) \qquad (3.5)$$

bits of information, which is the amount of information about $\tilde{x}_S$ contained in any $\tilde{y}_R$, given a value for $\tilde{x}_R$. (This bound is immediate from the definition of the information of an event as $\Sigma -p_i \log p_i$.) Side S may have to send more information than this if it is unable to code each $\tilde{x}_S$ value optimally. Its initial ignorance about the $\tilde{x}_R$ value will in general preclude optimal coding. For this

reason, only lower-bound results on information transmission can be obtained from the following derivations.

Since the values $\tilde{x}_R$ are indeterminate until the computation begins, the average amount of information transmitted during a computation is at least

$$\sum_{\tilde{x}_R} [\, p(\tilde{x}_R) \sum_{\tilde{y}_R} -p(\tilde{y}_R \mid \tilde{x}_R) \log p(\tilde{y}_R \mid \tilde{x}_R) \,], \tag{3.6}$$

where $p(\tilde{x}_R) = 1/|\{\tilde{x}_R\}|$ is the *a priori* probability of any particular value of $\tilde{x}_R$.

So far in the discussion, only one functional bisection has been considered, the one corresponding to the minimum bisection of the communication graph computing it. Many communication graphs can be designed to solve one problem, and each may generate a different functional bisection. It is necessary to consider all possible functional bisections in order to obtain a lower bound on the time required to compute a function on any communication graph.

Consider the minimum bisection of any communication graph solving a problem with N input values and K output values. A lower bound on the amount of information flow across the bisecting wires can be obtained as follows. A bisection is defined as splitting the source nodes in half, so that either $|\tilde{x}_R| = \lceil N/2 \rceil$ and $|\tilde{x}_S| = \lfloor N/2 \rfloor$, or $|\tilde{x}_S| = \lceil N/2 \rceil$ and $|\tilde{x}_R| = \lfloor N/2 \rfloor$. It also splits the sink nodes, so that one side or the other must compute at least $\lceil K/2 \rceil$ of the K output variables. Without loss of generality, assume that side R has this many output variables and let $\tilde{y}_R$ be any $\lceil K/2 \rceil$ of these, ignoring the rest of the problem outputs. If side R has less than half of the inputs, assign one additional input to it so that $|\tilde{x}_R| = \lceil N/2 \rceil$. (This possible addition of a component to $\tilde{x}_R$, and the possible omission of components from $\tilde{y}_R$, can only decrease the amount of information needed by side R from side S during its computation of $\tilde{y}_R$.)

These considerations lead to a *lower bound* on the *informational complexity* of a function, $H(F)$, defined as the minimum information flow across any bisection R, or

$$H(F) \geq \min_{R} \left[\sum_{\tilde{x}_R} p(\tilde{x}_R) \sum_{\tilde{y}_R} -p(\tilde{y}_R | \tilde{x}_R) \log p(\tilde{y}_R | \tilde{x}_R) \right] \quad (3.7)$$

where

$$|\tilde{x}_R| = \lceil N/2 \rceil, \ |\tilde{y}_R| = \lceil K/2 \rceil, \quad (3.8)$$

$$p(\tilde{x}_R) = 1/|\{\tilde{x}_R\}|, \quad (3.9)$$

and

$$p(\tilde{y}_R | \tilde{x}_R) = |\{\tilde{x}_S \text{ s.t. } F_R(\tilde{x}_R, \tilde{x}_S) = \tilde{y}_R\}| / |\{\tilde{x}_S\}|. \quad (3.10)$$

Lemma 5 summarizes the definitions and motivations of this section.

Lemma 5: The average time to compute a function F on a communication graph of width ω is at least $H(F)/\omega$.

Proof. At least H(F) bits must cross the minimum bisection of any communication graph computing F, during each evaluation of F. However, the bandwidth of the wires forming this bisection is limited to ω, so that an average evaluation of F takes at least $H(F)/\omega$ time. □

A definition of the worst-case information complexity of a function, $H_{worst}(F)$, may be made in a similar fashion. Referring to Equation (3.7), the inner sum may depend strongly on the value of $\tilde{x}_R$. The worst-case inputs in this formalism are those that have an $\tilde{x}_R$ that maximizes the inner sum. Thus

$$H_{worst}(F) \geq \min_{R} \max_{\tilde{x}_R} \sum_{\tilde{y}_R} -p(\tilde{y}_R | \tilde{x}_R) \log p(\tilde{y}_R | \tilde{x}_R), \quad (3.11)$$

where $|\tilde{x}_R|$, $|\tilde{y}_R|$, and $p(\tilde{y}_R | \tilde{x}_R)$ are given by Equations (3.8) and (3.10).

Equation (3.11) is not yet in its simplest form. The inner sum computes the entropy of a $\tilde{y}_R$ given an $\tilde{x}_R$, that is, the average word-length in bits of an optimal code describing $\tilde{y}_R$. In a worst-case analysis, it is the maximal word-length in the code for $\tilde{y}_R$ that is of interest, not the average length. Since there are $|\{\tilde{y}_R\}|$ different possible values for $\tilde{y}_R$, a signal at least $\log |\{\tilde{y}_R\}|$ bits long must be transmitted in the worst case. This gives a stronger bound on the worst-case informational complexity of a function,

$$H_{worst}(F) \geq \min_{R} \max_{\tilde{x}_R} \log |\{\tilde{y}_R\}|, \tag{3.12}$$

where

$$|\tilde{x}_R| = \lceil N/2 \rceil, \ |\tilde{y}_R| = \lceil K/2 \rceil,$$

and

$$\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S) \Big|_{\tilde{x}_R}. \tag{3.13}$$

Lemma 6: The worst-case time to compute a function F on a communication graph of width ω is at least $H_{worst}(F)/\omega$.

Proof. Any communication graph solving F will have to contend with a series of inputs for which $\tilde{x}_R$ and $\tilde{x}_S$ maximize the necessary information flow across its minimum bisection. Each of these "worst-case inputs" cannot be solved in an average time less than $H_{worst}(F)/\omega$, since the bandwidth across the bisection is at most ω. □

Examples. The informational complexity of some functions is trivial to evaluate. As noted previously, multiplication of a vector by a fixed scalar, $\tilde{y} = a\tilde{x}$, has zero complexity. In terms of Equation (3.7), choose R to be the obvious bisection that places corresponding elements of $\tilde{x}$ and $\tilde{y}$ on the same side. Then $p(\tilde{y}_R | \tilde{x}_R) = 0$ unless $\tilde{y}_R = \tilde{x}_R$, in which case $p(\tilde{y}_R | \tilde{x}_R) = 1$. In either case, the summand $p(\tilde{y}_R | \tilde{x}_R) \log p(\tilde{y}_R | \tilde{x}_R)$ is zero, so that zero information need cross the bisection R.

Two other simple functions of some interest are the comparison and equality functions. The comparison function takes two arguments ranging over the integers $\{0, 1, \ldots, M-1\}$ and produces one of two values as an output. The output is *1* if the first argument is greater than the second, *0* otherwise. The equality function is identical to the comparison function, except it returns *1* only if its arguments are equal.

Somewhat surprisingly, it is much easier to test for equality than to compare, judging by the information complexity of the two functions. Since there are two input variables, the only possible bisection puts one argument on either side. Let

the inputs be x_R and x_S, and let the boolean output variable be y_R. Assuming the inputs are uniformly and independently distributed, then $p(1|x_R) = 1/M$ and $p(0|x_R) = 1 - 1/M$ for the equality function. Thus

$$\begin{aligned} H(equality) &\geq \sum_{x_R} (1/M)[-(1/M)\log(1/M) - (1-1/M)\log(1-1/M)] \\ &\geq (1/M)\log M + (1 - 1/M)(\log e) \sum_{k\geq 1} 1/(kM^k) \\ &> (1/M)\log M. \end{aligned} \tag{3.14}$$

The comparison function has $p(1|x_R) = x_R/M$, $p(0|x_R) = 1 - x_R/M$, so that

$$\begin{aligned} H(compare) &\geq \sum_{x_R} (1/M)[-(x_R/M)\log(x_R/M) - (1-x_R/M)\log(1-x_R/M)] \\ &\geq 2\sum_{x_R} -(1/M)(x_R/M)\log(x_R/M) \\ &\geq 2\sum_{x_R \leq M/2} (1/M)(x_R/M) \\ &> 1/2. \end{aligned} \tag{3.15}$$

This analysis suggests that the equality function might be easier to compute than the comparison function, since the lower bound on its informational complexity is so much smaller. In fact, A. Yao [Yao 79] has proved that equality is indeed easier than comparison, in a similar model of computation. He shows that a signal of length $\theta(\log\log M)$ is necessary and sufficient to describe x_S so that an equality decision can be made with a vanishingly small error probability. The comparison function, on the other hand, needs a signal of $\theta(\log M)$ bits, which is of course the length of the obvious binary representation of x_S. (Yao's results underscore the fact that the informational complexity formulas developed in this section give only lower bounds on the necessary amount of information transmission. Fortunately, the lower bounds are nearly tight for Fourier transformation and sorting.)

A digression: Grigoryev's *l*-independence. The informational complexity of a function as defined in this thesis is similar to Grigoryev's definition [Grigoryev 76] of the *l*-independence of a function. A function

F mapping a vector of boolean variables $\tilde{x}$ onto a vector of boolean variables $\tilde{y}$ is l-independent if every partition A of F containing exactly l of the components of $\tilde{x}$ and $\tilde{y}$ satisfies

$$(\max_{\tilde{x}_A} |\{\tilde{y}_A\}|) \geq (2^{|\tilde{y}_A|-l} + 1), \tag{3.16}$$

where

$$|\tilde{x}_A| + |\tilde{y}_A| = l, \tag{3.17}$$

and $\tilde{y}_A$ is a subfunction of F, that is, a subset of the elements of $\tilde{y} = F(\tilde{x})$ when F is partially evaluated at $\tilde{x}_A$:

$$\tilde{y}_A \subseteq \{ F(\tilde{x}) \Big|_{\tilde{x}_A} \}. \tag{3.18}$$

Taking the logarithm of Equation (3.16) and replacing "for every partition A" by "minimum over all partitions A," Grigoryev's definition becomes

$$(\min_{A} \max_{\tilde{x}_A} \log |\{\tilde{y}_A\}|) > (|\tilde{y}_A| - 1). \tag{3.19}$$

Note that this form of Grigoryev's definition bears a strong resemblance to the worst-case informational complexity $H_{worst}(F)$ of Equation (3.12). There are two major differences, however. Grigoryev's definition covers only boolean functions, which are a special case ($M=2$) of Equation (3.12). More importantly, a partition A in an l-independence proof is not necessarily a bisection R in an informational complexity derivation. The bisections used in the definition of $H_{worst}(F)$ contain half of both input and output variables (Equation (3.13)), while a partition A contains a total of l variables. Grigoryev has made the more general definition, since the set of all partitions containing exactly $l = \lceil N/2 \rceil + \lceil K/2 \rceil$ variables is properly contained within the set of all bisections of N input and K output variables. A function $F:\{0,1\}^N \leftarrow \{0,1\}^K$ that is $(\lceil N/2 \rceil + \lceil K/2 \rceil)$-independent must have worst-case informational complexity of at least $\lceil K/2 \rceil$, since its "best bisection" can require no less information flow than its "best partition." As will be seen later in this chapter, a funciton with informational complexity H has an area-time tradeoff of $AT^2 = \Omega(H^2)$. Thus any chip that evaluates an N-independent function in worst-case time T and area A must obey $AT^2 = \Omega(N^2 \log^2 N)$.

3.3.1 Discrete Fourier transformation

Consider the reduced DFT computation in which the communication graph is to evaluate only the first $\lfloor N/2 \rfloor$ of the N outputs. Furthermore, it is to produce only the *mod* Q residues of these outputs, where Q is any prime dividing the ring modulus M.

Following the definition in Section 1.2.1, the reduced DFT is a matrix vector multiplication

$$\tilde{y}_r = A_r \tilde{x} \bmod Q, \tag{3.20}$$

where the elements of A_r are given by

$$A_r[i,j] = \alpha^{ij} \quad \text{for } 0 \le i < \lfloor N/2 \rfloor,\ 0 \le j < N, \tag{3.21}$$

and α is a principal N-th root of unity in the ring of multiplication and addition modulo M. The ring modulus M must satisfy

$$M = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k}, \tag{3.22}$$

where

$$N \mid gcd(p_1 - 1, p_2 - 1, \ldots, p_k - 1).$$

Since Q is one of the prime factors of M, the entire computation can be done $mod\ Q$. The reduced DFT is thus

$$\tilde{y}_r = A_r(\tilde{x} \bmod Q), \tag{3.23}$$

where

$$A_r[i,j] = (\alpha_Q)^{ij} \text{ for } 0 \le i < \lfloor N/2 \rfloor,\ 0 \le j < N, \tag{3.24}$$

and

$$\alpha_Q = \alpha \bmod Q. \tag{3.25}$$

All the α_Q are distinct [Agarwal 75].

The probability of any particular value of $\tilde{x}\ mod\ Q$ is simple to derive. Each $mod\ Q$ residue arises in $\{0, 1, \ldots, M-1\}$ exactly M/Q times, so that

$$p(\tilde{x} \bmod Q) = (M/Q)^N p(\tilde{x}) = 1/Q^N. \tag{3.26}$$

The reduced DFT can thus be considered to be a mapping from $\{0, 1, \ldots, Q-1\}^N$ onto $\{0, 1, \ldots, Q-1\}^{\lfloor N/2 \rfloor}$, each point in the domain having equal likelihood. Assumption L4 is thus satisfied for the reduced DFT. Furthermore any computation of a DFT is also a computation of a reduced DFT, in the sense formalized below. Lower bounds for the reduced DFT thus apply immediately to the full DFT.

Lemma 7: Any communication graph that solves a full DFT also solves a reduced DFT in the same amount of time.

Proof. According to assumption L7, the original output assertion for each of the N outputs is satisfied at the end of a DFT computation. In other words, the final state of the sink nodes computing one of these outputs corresponds to a particular output value in the full ring of modulo M arithmetic. Each final state of a sink node also corresponds to a particular value in the reduced ring of modulo Q arithmetic. Thus a new set of output assertions can be written to map final sink node states onto correct output values for a reduced DFT computation, leaving the rest of the communication graph intact. □

The importance of the reduced DFT is that any bisection of domain elements still spans its entire range. The full DFT does not enjoy this property.

Lemma 8: [Tompa 78, Valiant 76]: The matrix formed by the selection of any $\lfloor N/2 \rfloor$ columns of A_r is invertible.

Proof. Let $\{b_j\}$ be the indices of the selected columns. The resulting matrix B is then

$$B[i,j] = (\alpha_Q)^{ib_j} \quad for\ 0 \le i,j < \lfloor N/2 \rfloor,\ 0 \le b_j < N. \qquad (3.27)$$

The transpose of B is a Vandermonde matrix in which row j has the first $\lfloor N/2 \rfloor$ powers of $(\alpha_Q)^{b_j}$. The base elements of each row, $(\alpha_Q)^{b_j}$, are distinct because they are selected from the necessarily distinct powers of α_Q. The determinant of B^T is thus non-zero. Since Q is prime, integer arithmetic *mod* Q forms a field, and the determinant of B^T has a unique multiplicative inverse. Therefore, B is invertible. □

The preceding lemma is not quite powerful enough to evaluate the information complexity of the reduced DFT. It shows that if all of the first $\lfloor N/2 \rfloor$ outputs were on one side of a bisected communication graph, they would need complete information about any $\lfloor N/2 \rfloor$ of the inputs on the far side. In general, one can only expect to find a fraction of those outputs on one side, so the following lemma is of use.

Lemma 9: If $\tilde{y} = F(\tilde{x})$ is a bijection, $F:\{0, 1, \ldots, Q-1\}^k \rightarrow \{0, 1, \ldots, Q-1\}^k$, and if all elements of the domain $\tilde{x}$ are equally likely, then for any selection $\{C\}$ of $|\tilde{y}_C|$ components of $\tilde{y}$,

$$\sum_{\tilde{y}_C} -p(\tilde{y}_C) \log p(\tilde{y}_C) = |\tilde{y}_C| \log Q. \tag{3.28}$$

Proof. The probability, $p(\tilde{y}_C)$, of obtaining a particular value $\tilde{y}_C$ for the selected set of components can be evaluated by considering all possible values for $\tilde{x}$,

$$p(\tilde{y}_C) = \sum_{\tilde{x}} \delta[\tilde{y}_C = F_C(\tilde{x})]\, p(\tilde{x}). \tag{3.29}$$

The "delta function," $\delta[\tilde{y}_C = F_C(\tilde{x})]$, is one if the projection of $F(\tilde{x})$ onto the variables in C equals $\tilde{y}_C$; otherwise $\delta[...]$ is zero. The probability $p(\tilde{x})$ of any particular $\tilde{x}$ is $1/Q^k$, since all $\tilde{x}$ are equally likely. Also, every distinct $\tilde{x}$ gives a distinct $\tilde{y}$, because F is a bijection. There are just $k-|\tilde{y}_C|$ components in $\tilde{y}$ outside of $\tilde{y}_C$, so there are at most $Q^{k-|\tilde{y}_C|}$ distinct values of $\tilde{y}$ for fixed $\tilde{y}_C$. Thus at most $Q^{k-|y_C|}$ distinct $\tilde{x}$ can give $F_C(\tilde{x}) = \tilde{y}_C$. These considerations bound the size of the summand in Equation (3.29), giving

$$\begin{aligned} p(\tilde{y}_C) &\leq (Q^{k-|\tilde{y}_C|})(1/Q^k) \\ &\leq 1/Q^{|\tilde{y}_C|}. \end{aligned} \tag{3.30}$$

This inequality must be sharp for all $\tilde{y}_C$, by the following argument. The sum of $p(\tilde{y}_C)$ over all $\tilde{y}_C$ must be unity, as they are probabilities. There are exactly $Q^{|\tilde{y}_C|}$ distinct $\tilde{y}_C$, so

$$\operatorname*{avg}_{\tilde{y}_C} p(\tilde{y}_C) = 1/Q^{|\tilde{y}_C|}. \tag{3.31}$$

If any of the $p(\tilde{y}_C)$ were less than $1/Q^{|\tilde{y}_C|}$, the average would fall below this figure. Thus

$$p(\tilde{y}_C) = 1/Q^{|\tilde{y}_C|}. \tag{3.32}$$

The lemma is now immediate.

$$\sum_{\tilde{y}_C} -p(\tilde{y}_C)\log p(\tilde{y}_C) = Q^{|\tilde{y}_C|}(-1/Q^{|\tilde{y}_C|})\log(1/Q^{|\tilde{y}_C|})$$

$$= |\tilde{y}_C| \log Q. \qquad \square \qquad (3.33)$$

Finally, all the groundwork is in place for the main theorem of this section.

Theorem 10: The average time T to compute an N-element DFT on a communication graph with minimum bisection width ω is bounded by $T > (\lfloor N/4 \rfloor \log N)/\omega$.

Proof. By Lemma 7, the time taken to compute a DFT is bounded from below by the time taken to compute a reduced DFT. The informational complexity of a reduced DFT is, by Equation (3.7),

$$H(DFT) \geq \min_R \Big[\sum_{\tilde{x}_R} p(\tilde{x}_R) \sum_{\tilde{y}_{rR}} -p(\tilde{y}_{rR} \mid \tilde{x}_R) \log p(\tilde{y}_{rR} \mid \tilde{x}_R) \Big], \qquad (3.34)$$

where

$$|\tilde{x}_R| = \lceil N/2 \rceil \text{ and } |\tilde{y}_{rR}| = \lceil \lfloor N/2 \rfloor / 2 \rceil, \qquad (3.35)$$

since the reduced DFT has N inputs and $\lfloor N/2 \rfloor$ outputs.

For each bisection R, the reduced DFT may be expressed as

$$\tilde{y}_r = A_{rR}\,\tilde{x}_R + A_{rS}\,\tilde{x}_S, \qquad (3.36)$$

where A_{rR} is formed from the reduced DFT matrix by selecting the columns whose variables are in R. Similarly, the matrix A_{rS} is formed from the $\lfloor N/2 \rfloor$ columns of A_r not in R.

By Lemma 8, A_{rS} is invertible, so that for fixed $\tilde{x}_R$,

$$\tilde{y}_r = F_{\tilde{x}_R}(\tilde{x}_S) \qquad (3.37)$$

is a bijection from $\{0, 1, \ldots, Q-1\}^{\lfloor N/2 \rfloor}$ onto $\{0, 1, \ldots, Q-1\}^{\lfloor N/2 \rfloor}$.

Lemma 9 can now be applied to the expression for $H(DFT)$ above, using

$$\tilde{y} = \tilde{y}_r, \quad \tilde{x} = \tilde{x}_S, \quad F(\tilde{x}) = F_R(\tilde{x})\Big|_{\tilde{x}_R}, \qquad (3.38)$$

$$k = \lfloor N/2 \rfloor, \ \tilde{y}_C = \tilde{y}_{rR}, \ |\tilde{y}_C| = \lceil k/2 \rceil = \lceil \lfloor N/2 \rfloor / 2 \rceil, \qquad (3.39)$$

and

$$p(\tilde{y}_C) = p(\tilde{y}_{rR} \mid \tilde{x}_R), \tag{3.40}$$

so that

$$\begin{aligned} H(DFT) &\geq \min_R \Big[\sum_{\tilde{x}_R} p(\tilde{x}_R) (\lceil \lfloor N/2 \rfloor / 2 \rceil \log Q) \Big] \\ &\geq (\lceil \lfloor N/2 \rfloor / 2 \rceil \log Q) \min_R \sum_{\tilde{x}_R} p(\tilde{x}_R) \\ &\geq \lceil \lfloor N/2 \rfloor / 2 \rceil \log Q. \end{aligned} \tag{3.41}$$

Since any prime Q dividing the modulus M of a DFT must be greater than N (see Section 1.2.1),

$$H(DFT) > \lfloor N/4 \rfloor \log N. \tag{3.42}$$

By Lemma 5, the average time T taken to solve a DFT on a communication graph with width ω is bounded by

$$T \geq H(DFT)/\omega \tag{3.43}$$

so that

$$T > (\lfloor N/4 \rfloor \log N)/\omega. \ \square \tag{3.44}$$

Corollary 11: The performance of any communication graph with area A that solves a DFT in average time T is limited by $AT^2 \geq \lfloor N/8 \rfloor^2 \log^2 N$.

Proof. By Theorem 2, the area of any graph with bisection width ω is bounded by $A > \omega^2/4$. Squaring Equation (3.44) gives $T^2 \geq \lfloor N/4 \rfloor^2 \log^2 N/\omega^2$, hence the Theorem. $\square$

Corollary 12: The relation $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$ is satisfied by any communication graph with area A that takes average time T to solve an N-element DFT, for all x such that $0 \leq x \leq 1$.

Proof. The area of any communication graph with N source nodes must be $\Omega(N)$, since each source node takes up at least one unit of area. By Theorem 2 and Theorem 10,

$$\begin{aligned} AT^{2x} &= \Omega((N + \omega^2/4)((N \log N)/4\omega)^{2x}) \\ &= \Omega(N^{1+2x} (\log^{2x} N) \omega^{-2x} + N^{2x} (\log^{2x} N) \omega^{2-2x}). \end{aligned} \tag{3.45}$$

The value of AT^{2x} can be minimized with respect to ωi since the first term decreases with ω and the second term increases with ω, for $0 < x < 1$. Equating the derivative of Equation (3.45) to zero gives

$$d/d\omega\,(N^{1+2x}(\log^{2x}N)\omega^{-2x} + N^{2x}(\log^{2x}N)\omega^{2-2x}) = 0, \tag{3.46}$$

$$-2xN^{1+2x}(\log^{2x}N)\omega^{-2x-1} + (2-2x)N^{2x}(\log^{2x}N)\omega^{1-2x} = 0, \tag{3.47}$$

$$(\omega^{1-2x})/(\omega^{-2x-1}) = 2xN^{1+2x}(\log^{2x}N)/((2-2x)N^{2x}\log^{2x}N), \tag{3.48}$$

$$\omega^2 = Nx/(1-x), \tag{3.49}$$

so that

$$\omega = \theta(N^{1/2}) \tag{3.50}$$

leads to the best possible lower bound on AT^{2x} performance for $0 < x < 1$. Specifically, when this value is used for ω, Equation (3.45) becomes

$$AT^{2x} = \Omega(N^{1+x}\log^{2x}N), \tag{3.51}$$

proving the Corollary for $0 < x < 1$. Theorem 11 covers the case $x=1$, and the case $x=0$ is immediate. □

Corollary 13: At least $\Omega(N^{3/2}\log N)$ units of energy must be dissipated by any chip solving an N point DFT.

Proof. A communication graph can be drawn for any chip solving a DFT, using the correspondence scheme of Section 2.3. By Corollary 12, the product of area with time for such a chip must be at least $\Omega(N^{3/2}\log N)$. As indicated in Section 1.1, the product of area with time measures energy. □

3.3.2 Sorting

The average informational complexity of the sorting problem appears to be difficult to evaluate, but a worst-case bound is almost immediate. Recall that the sorting problem was defined in Section 1.2.2 as an N-input, N-output problem. The inputs are chosen from the integers $\{0, 1, \ldots, M-1\}$. The outputs are to be a permutation of the inputs into non-decreasing order. Note that the size of M is extremely important. If $M = 2$, then each half of a bisected sorting circuit needs to know only the total number of zero input values on the other side of the bisector.

The informational complexity of sorting, for $M = 2$, is thus only $O(\log N)$. The rest of this chapter makes liberal use of the assumption (L4) that $M = N^{1+\varepsilon}$ for some positive ε.

A bound on the informational complexity of the sorting problem is obtained through an analysis of a "reduced" version, analogous to the reduced DFT of the previous subsection. Define the reduced sorting problem as a sort in which only the first $\lfloor N/2 \rfloor$ output values are required to be correct. Call this problem $\tilde{y} = RSORT(\tilde{x})$, where $|\tilde{y}| = \lfloor N/2 \rfloor$ and $|\tilde{x}| = N$.

Any communication graph solving the sorting problem must also solve RSORT: consider a reduced graph from which the assertions associated with the last $\lceil N/2 \rceil$ output variables have been deleted. The time required by an *RSORT* is thus a lower bound on sorting time. The following lemma bounds the informational complexity of *RSORT*.

Lemma 14: $H_{worst}(RSORT) > \lfloor N/4 \rfloor \log(2M/N)$.

Proof. From Equation (3.12),

$$H_{worst}(RSORT) = \min_{R} \max_{\tilde{x}_R} \log |\{\tilde{y}_R\}|, \tag{3.52}$$

where

$$|\tilde{x}_R| = \lceil N/2 \rceil \text{ and } |\tilde{y}_R| = \lceil \lfloor N/2 \rfloor / 2 \rceil. \tag{3.53}$$

The variability in the outputs of side R, $|\{\tilde{y}_R\}|$, is maximized when the elements of $\tilde{x}_R$ are all equal to $M-1$. In this case, the outputs of *RSORT* are only dependent on $\tilde{x}_S$, for this vector must contain all of the $\lfloor N/2 \rfloor$ smallest input values. As $\tilde{x}_S$ varies over all its possibilities, so will $\tilde{y}_R$, so that each of the $\lceil \lfloor N/2 \rfloor / 2 \rceil$ elements of $\tilde{y}_R$ can take on any value in $\{0, 1, \ldots, M-1\}$. Since $\tilde{y}_R$ is in sorted order, there are not quite $M^{\lceil \lfloor N/2 \rfloor / 2 \rceil}$ possible values for $\tilde{y}_R$. Instead,

$$|\{\tilde{y}_R\}| = \binom{M + \lceil \lfloor N/2 \rfloor / 2 \rceil - 1}{\lceil \lfloor N/2 \rfloor / 2 \rceil}, \tag{3.54}$$

the number of selections of $\lceil \lfloor N/2 \rfloor / 2 \rceil$ indistinct elements from M possibilities. If the elements in each selection are arranged in increasing order, the selections can be seen to be in one-to-one correspondence with the $\tilde{y}_R$.

Thus,

$$H_{worst}(RSORT) \geq \log |\{\tilde{y}_R\}| \tag{3.55}$$

$$> \log (M^{\lceil \lfloor N/2 \rfloor /2 \rceil} / \lceil \lfloor N/2 \rfloor /2 \rceil^{\lceil \lfloor N/2 \rfloor /2 \rceil}) \tag{3.56}$$

$$> \lceil \lfloor N/2 \rfloor /2 \rceil \log (M / \lceil \lfloor N/2 \rfloor /2 \rceil). \tag{3.57}$$

Applying the inequalities

$$N/2 \geq \lceil \lfloor N/2 \rfloor /2 \rceil \geq \lfloor N/4 \rfloor \tag{3.58}$$

to Equation (3.57) gives

$$H_{worst}(RSORT) > \lfloor N/4 \rfloor \log (2M/N). \quad \square \tag{3.59}$$

Given this bound on the worst-case information complexity of *RSORT*, the following performance bounds are immediate for the full sorting problem.

> **Theorem 15:** The worst-case time taken by any communication graph of width ω to sort N numbers chosen from $\{0, 1, \ldots, M-1\}$ is at least $\Omega(N \log N)/\omega$, if $M = N^{1+\varepsilon}$ for some fixed $\varepsilon > 0$.

Proof. By Lemma 6, the worst-case time to perform an *RSORT* is at least $H_{worst}(RSORT)/\omega$; from Equation (3.57), this is $\Omega(N \log N)/\omega$ since $M = N^{1+\varepsilon}$. Any communication graph that sorts also performs an *RSORT*, so the graph must take at least this much time to produce its results. □

> **Corollary 16:** The relation $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$ is satisfied by any communication graph with area A that takes worst-case time T to solve a sorting problem of size N, for all x such that $0 \leq x \leq 1$.

Proof. Identical to the proof of Corollary 12. □

> **Corollary 17:** It takes at least $\Omega(N^{3/2} \log N)$ units of energy to sort N numbers on any VLSI chip.

Proof. Identical to the proof of Corollary 13. □

Chapter 3 is now complete. In summary, the "informational complexity" of a function is defined as the minimum necessary information flow across any bisection of a communication graph computing that function. Also by definition, the available bandwidth across any bisection of a communication graph is proportional

to its "minimum bisection width." Thus a lower bound on the time needed to compute a function is its informational complexity divided by the minimum bisection width of the graph computing it. Theorems 10 and 15 evaluate this lower bound for the functions of sorting and Fourier transformation. Previously (Theorem 2), the area of a communication graph was shown to be at least $\omega^2/4$, where ω is its minimum bisection width. Corollaries 12 and 16 combine these lower bounds on area and time to give the general result that $AT^{2x} = \Omega(N^{1+x}log^{2x}N)$ for $0 \leq x \leq 1$.

Chapter 4
Upper bounds

An upper bound in the VLSI model of computation is proved by the existence of an admissible communication graph achieving a given performance level. This chapter details four constructions, two solving the Fourier transform problem and two solving the sorting problem. Of the two graphs solving each problem, one is based on the shuffle-exchange interconnection pattern, and one is based on the square mesh. The shuffle-exchange designs are fast but large: the Fourier transform circuit operates in time $T = O(\log^2 N)$ and area $A = O(N^2/\log^{1/2} N)$; the sorting circuit has $T = O(\log^3 N)$ and the same area. The mesh designs for both problems are relatively slow and small, $T = O(N^{1/2} \log\log N)$ and $A = O(N \log^2 N)$. All four designs are nearly optimal under the AT^2 metric, and the mesh designs are nearly optimal under any AT^{2x} metric, for $0 \leq x \leq 1$.

The constructions are described in a top-down fashion, starting with a discussion of the basic algorithms, the bitonic sort and the fast Fourier transform (FFT). In Section 4.1, these algorithms are defined as networks consisting of $O(N \log N)$ or more *cells* each capable of performing a two-element sort or a two-element Fourier transform. Section 4.2 discusses the way in which a network construction leads to a *recirculation algorithm* that uses only $O(N)$ cells. Section 4.3 gives two VLSI implementations of the FFT algorithm, one for a mesh-based recirculation network, one for a shuffle-exchange-based recirculation network. These two networks are used to implement the bitonic sorting algorithm in Section 4.4. Section 4.5 develops approximations to the actual size and speed of the VLSI implementations of sorting and FFT circuits, comparing these to their asymptotic performance measures.

4.1 Algorithms

Both the bitonic sort and the fast Fourier transform are often described as large networks of fairly simple cells. The FFT network will be discussed first, since its construction is the simpler of the two.

An N-element FFT network [Cochran 67] is built from a total of $\frac{1}{2}N \log N$ multiply-add cells. Figures 4-1 and 4-2 illustrate the recursive construction of an N-element network from $N/2$ cells and two $N/2$-element networks. Figure 4-3 is the 8-element FFT network that results from this method of construction.

The basis of the recursive construction is the two-element FFT network composed of a single multiply-add cell. A cell marked k produces as outputs

$$Y_0 = X_0 + \alpha^k X_1 \qquad (4.1)$$

and

$$Y_1 = X_0 - \alpha^k X_1, \qquad (4.2)$$

where X_0 and X_1 are its two input values (here α is an N-*th* root of unity, as defined in Subsection 1.2.1). The exponent of α is parameterized since different cells have different values for k. The construction of Figure 4-2 defines the way in which cells are assigned values for their parameter k. In this construction, networks and subnetworks are also parameterized: a *(k)*-FFT network is built from an $N/2$-element *(0)*-FFT network and an $N/2$-element *(N/4)*-FFT network.

The outputs of an N-element (0)-FFT network are the N outputs of the discrete Fourier transform. However, these outputs appear in "bit-reversed" order [Cochran 67]. That is, if the output indices are expressed as *(log N)*-bit binary numbers, then each output appears in the position denoted by reading its index in reversed order. For example, the fifth output of an 8-element FFT, Y_4, appears in position $(001)_2 = 1$, since the binary representation of its index is $(100)_2$.

It is now time to examine sorting networks. The N-element bitonic sorting network [Batcher 68] is built of $\frac{1}{2}N(\log N)(\log N + 1)$ comparison-exchange cells,

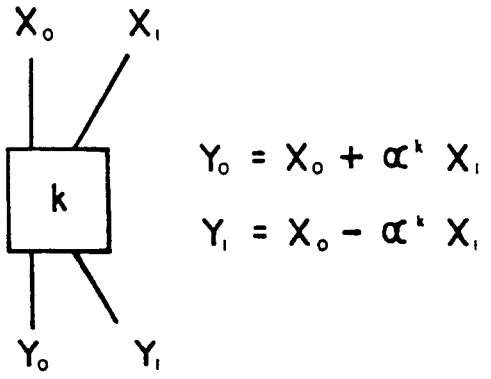

Figure 4-1: Multiply-add cell or 2-element FFT network.

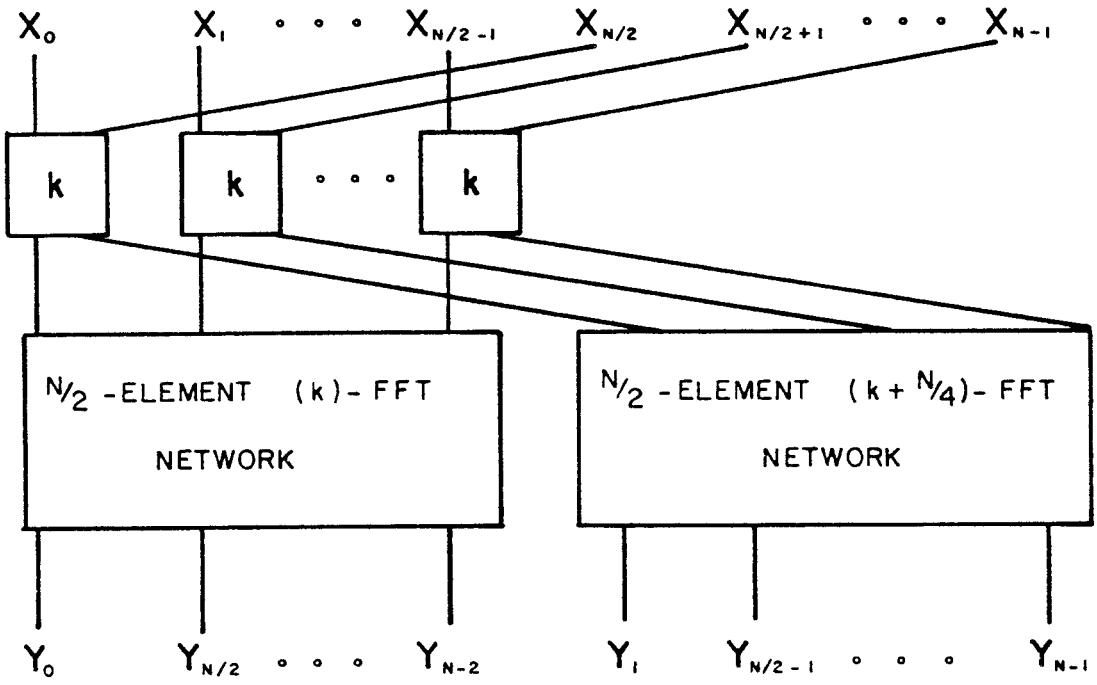

Figure 4-2: Recursive construction of an *N*-element *(k)*-FFT network from ½*N(log N)* multiply-add cells.

interconnected as shown in Figures 4-4 through 4-7. A comparison-exchange cell is a two-input, two-output device that compares the values of its two inputs, sending the larger toward the point of its arrow. As indicated in Figure 4-6, a sorting network is defined in a doubly-recursive fashion, from two smaller sorting networks and one bitonic merger. Note that a bitonic merger is topologically identical to the FFT network of Figure 4-2, since it has the same recursive construction, from *N/2* cells and two half-sized networks.

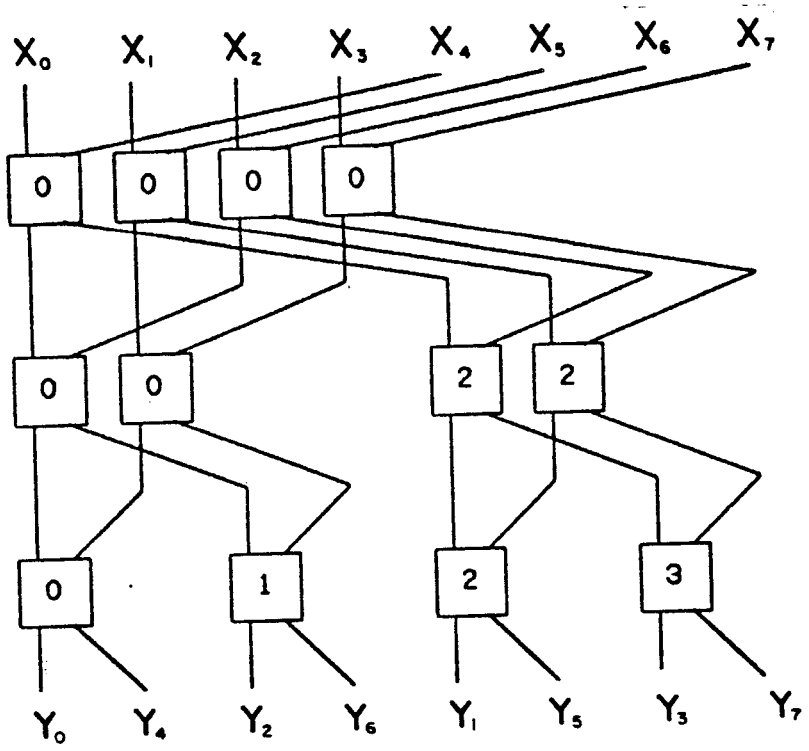

Figure 4-3: An *8*-element FFT network.

The term bitonic seems to have been coined by Batcher [Batcher 68], by analogy with monotonic. A bitonic sequence is the concatenation of two monotonic sequences, one increasing and the other decreasing. In the article cited above, Batcher also makes a distinction between a "bitonic sorter" (his name for a bitonic merger) and the complete bitonic sorting network. In an effort to avoid confusion between these two networks, Batcher's distinction is *not* observed here.

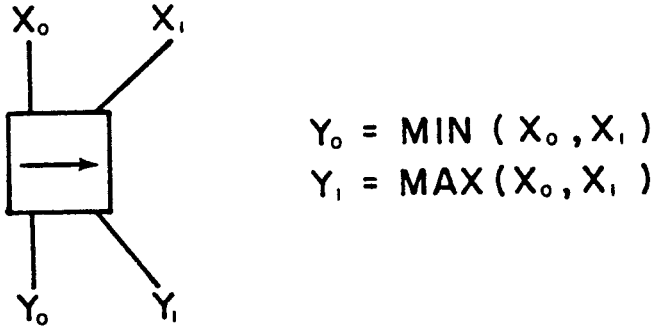

Figure 4-4: Comparison-exchange cell or *2*-element sorting network.

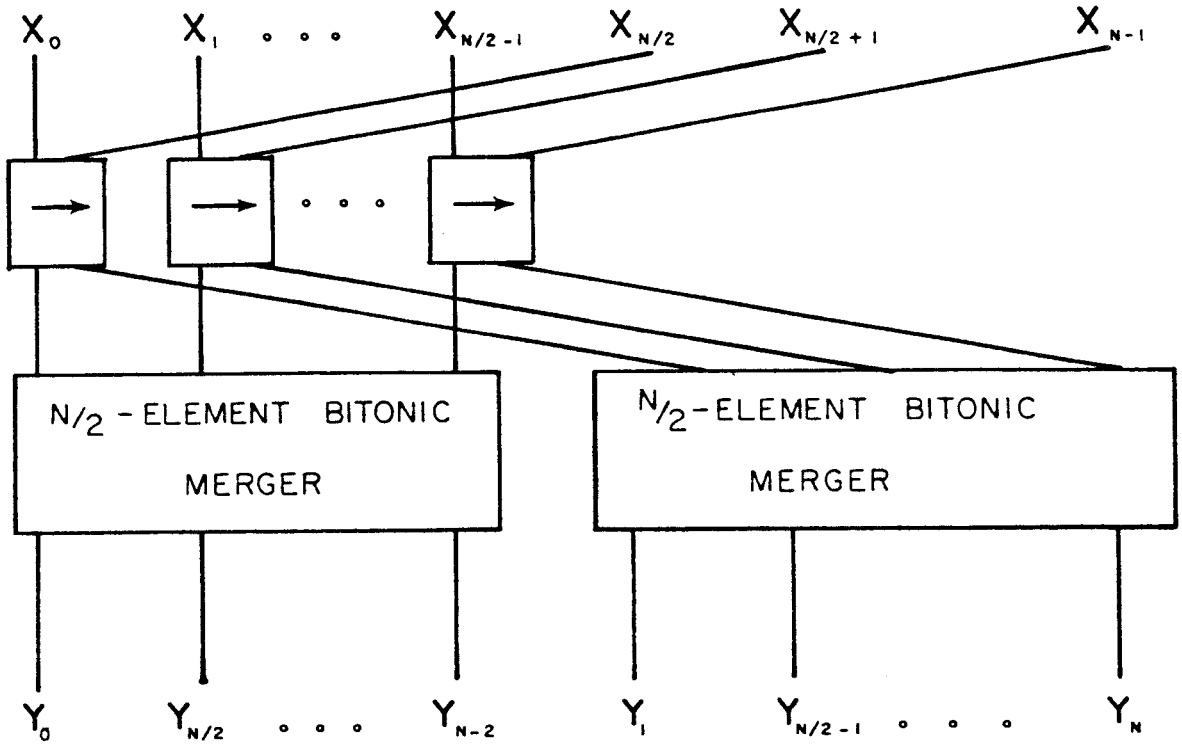

Figure 4-5: Recursive construction of an *N*-element bitonic merger.

The bases for the recursions of Figures 4-5 and 4-6 are formed trivially by the single comparison-exchange cell of Figure 4-4, which may be considered either a two-element bitonic merger or a two-element sorting network. The arrow inside a sorting network indicate the "direction" of its sorting process: the larger input elements move toward the point of the arrow, following Knuth's notation ([Knuth 73], p. 237). The direction of these arrows is critical in the construction, for one of the two *N/2*-element sorting networks of Figure 4-6 must produce its outputs in ascending order, while the other must produce a descending sequence.

The action of a sorting network can be described in an intuitive fashion [Stone 71]. The graph of the outputs of the left-hand *N/2*-element sorting network may be visualized as /, for its rightmost outputs are the largest. Similarly, the graph of the outputs of the right-hand *N/2*-element sorting network looks like \. The inputs to the *N*-element bitonic merger are thus a bitonic sequence $\wedge$. Just inside of the *N*-element bitonic merger, the two arms of the $\wedge$ are interleaved to form an $\times$. The

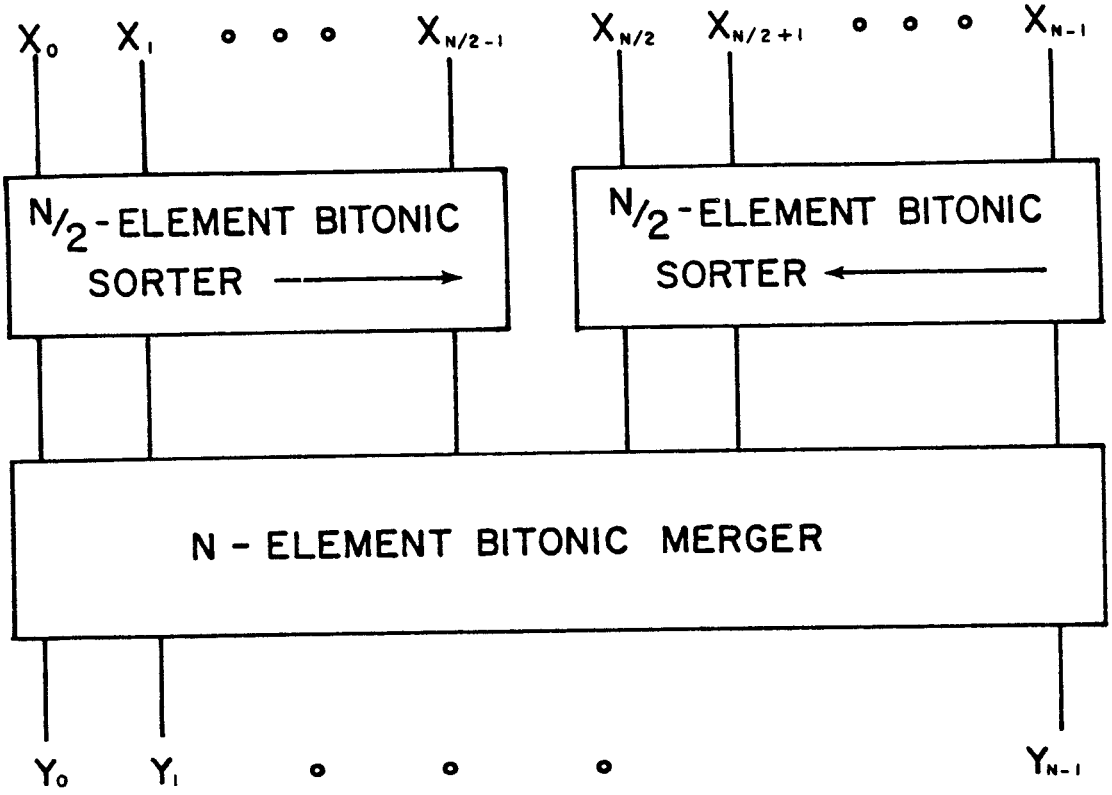

Figure 4-6: Construction of an *N*-element bitonic sorting network from two *N/2*-element bitonic sorting networks and one *N*-element bitonic merger.

comparison-exchange cells are able to pick apart the top and bottom halves of this $\times$ into a $\wedge$ and a $\vee$, both of which are bitonic sequences. The $\wedge$ is passed to the left and the $\vee$ is sent to the right. Since all elements of the $\wedge$ are less than any element of the $\vee$, the outputs of the two *N/2*-element bitonic mergers look like / and / and can be immediately combined into a fully sorted sequence /.

Note that there are two distinct ways of forming an ascending sorting network ($\rightarrow$) from a descending sorting network ($\leftarrow$). One way is to "flip" the network along a vertical axis, so that the rightmost and largest output becomes the leftmost output. This is the approach used on page 237 of [Knuth 73]. The other way, chosen by Stone [Stone 71] and adopted here, is to reverse the directions of all the comparison-exchange cells inside a descending sorting network. Figure 4-7 shows the eight-element sorting network obtained from the construction of Figure 4-6, using this

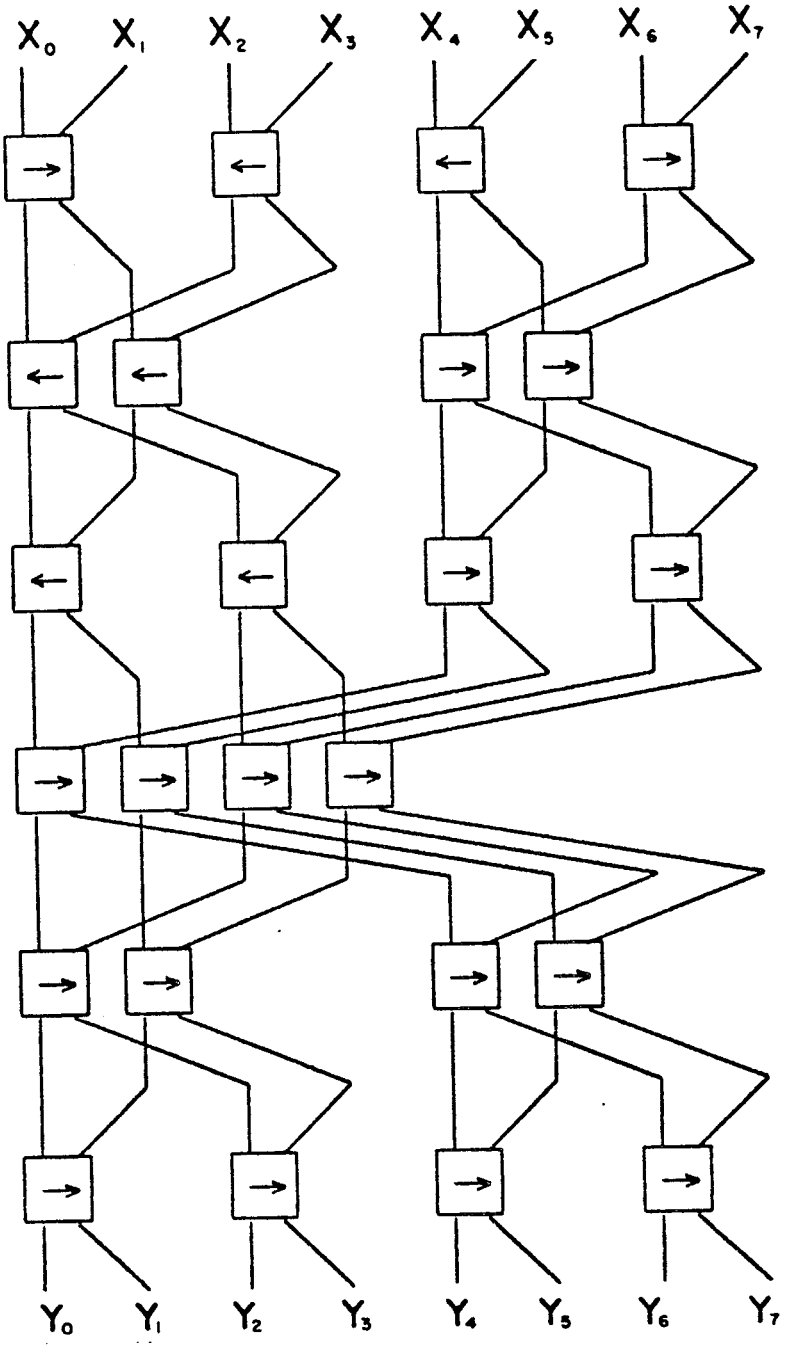

Figure 4-7: An 8-element bitonic sorting network.

type of descending sorter. The two methods of reversal give distinct networks. In particular, the top row of an eight-element sorting network built using Knuth's approach has comparison-exchange cell directions ($\rightarrow \leftarrow \rightarrow \leftarrow$), while the top row of Figure 4-7 is ($\rightarrow \leftarrow \leftarrow \rightarrow$). The virtue of the latter approach is that it has a more

obvious mapping onto the shuffle-exchange pattern, as will be seen in Subsection 4.4.1.

4.2 Recirculation algorithms

The sorting and FFT networks described in the previous section could be implemented directly in VLSI, although this would be grossly inefficient. During the course of a computation, each cell is active only once, producing just one pair of results.

There are two ways to boost the efficiency of sorting and FFT networks, *pipelining* and *recirculation*. The pipelining approach introduces a row of registers between each row of cells in Figures 4-3 and 4-7. A new problem can be fed into the top row of the network as soon as the previous problem inputs have emerged from that row. Since a pipelined design solves many problems simultaneously, it can solve a series of problems more quickly than a non-pipelined design. However, the pipelined design is no faster at solving a single problem, and thus does not have a better time performance under the ground rules of this thesis. (An interesting extension to the VLSI model of computation would explore different notions of time performance. For example, solution time might be defined as the time between successve problem solutions, under the assumption that the circuit is always being fed with new problems.)

The second method of increasing efficiency, *recirculation*, uses one row of cells many times during the solution of a single problem. During each stage of the computation, this row simulates the action of one of the rows in the complete network. The inputs needed by the row as it simulates the $(k+1)st$ row of the network are obtained from the outputs of the kth stage of computation. A *recirculation network* is used to handle this data flow. The structure of this network is important. If its connectivity is not precisely right, the data will have to circulate more than once through the row before it reaches the correct cell. As shown in Subsections 4.3.3 and 4.4.1, the shuffle-exchange pattern is an ideal recirculation

network: one pass through the network is sufficient between each stage of a FFT or bitonic sort algorithm. The mesh pattern is not nearly so well suited to these algorithms, but it does require much less silicon area, as shown later in this chapter.

4.3 VLSI implementations of the FFT

This section is divided into four parts. The first part shows how an N-element FFT can be performed on a row of N *multiply-add* cells, using a mesh-type recirculation network. The resulting algorithm is essentially identical to one proposed for the the Illiac IV computer [Stevens 71].

The second subsection examines the problem of implementing a multiply-add cell in VLSI. A construction is proposed, and its area-time performance is analyzed. This leads to area and time results for a complete mesh-based FFT circuit.

The construction of Subsection 4.3.3 uses a different recirculation network based on the shuffle-exchange graph. Area and time results for this FFT circuit are developed in Subsection 4.3.4, after the problem of embedding the shuffle-exchange graph in the plane has been solved.

4.3.1 Performing the FFT on a mesh

The square mesh interconnection pattern takes its name from its appearance when drawn in its most compact form, as in Figure 4-8. In a mesh with $N = n^2$ cells, an interior cell i connects "horizontally" to cells $i-1$ and $i+1$, and "vertically" to cells $i-n$ and $i+n$. Note that there are no end-around connections, so that nodes on the periphery of the mesh have fewer than four neighbors. Figure 4-8 shows the row-major indexing scheme for the cells in a mesh that is implicit in the definition given above for cell connectivity.

The precise functionality of the cells in the mesh will be developed in the next subsection, in keeping with a top-down approach to the description of the

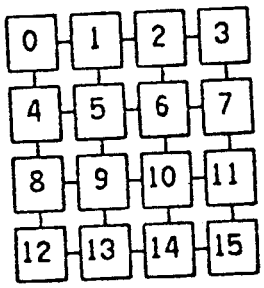

Figure 4-8: A 16-cell mesh drawn in its most compact form.

constructions. At the present level of detail, cells may be thought of as message-driven processors. Each cell can *create* a new message from the information contained in its registers, *addressing* it to any other cell in the network. Each cell can also *forward* messages along the shortest path to their destinations. Finally, each cell can *receive* messages addressed to it, and update its registers appropriately.

To understand the way in which an FFT can be performed on a mesh, it is best to visualize the mesh in the linearized form of Figure 4-9. In this representation, the horizontal connections are still quite short, but the vertical connections have been lengthened dramatically (they must skip over $N^{1/2}-1$ intervening cells). Of course, the VLSI circuit will not be laid out in this fashion, as the vertical connections would waste a lot of area.

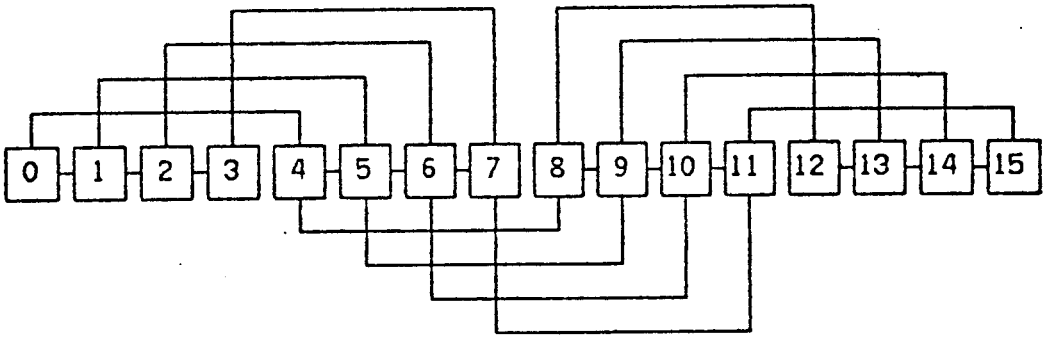

Figure 4-9: A 16-cell mesh drawn in linear form.

There is a natural correspondence between the linear form of the N-cell mesh and the N-element FFT network. Each of the N columns of cells in the FFT network corresponds to one of the N cells in the mesh; the interconnections between rows of cells in the FFT network correspond to data movement among the cells in the mesh (a "recirculation"). The comments made so far apply to any recirculation algorithm on any recirculation network: what makes the N-cell mesh correspond nicely to the N-element FFT network is the fact that each recirculation can be mapped efficiently onto either the horizontal or vertical interconnections of the mesh. Referring to the *16*-element FFT network of Figure 4-10, the interconnection pattern between each pair of rows consists of connections between columns whose indices differ by a power of *2*. Similarly, in Figure 4-9, connections exist between cells whose indices differ by a power of *2*.

The actions performed by cell *0* of a *16*-cell mesh performing a *16*-element FFT can be described with reference to Figures 4-9 and 4-10. Initially, the *16* inputs to the FFT are stored one per cell: cell i contains input X_i. Looking at the top of column *0* of the FFT network, cell *0* receives input X_8 from cell *8*, and does a multiply-add step with coefficient α_0 on this new value and its original input X_0. It keeps one of its two outputs, sending the other back to cell *8*. Moving to the second row of cells in the FFT network, cell *0* receives a value from cell *4*, does a multiply-add step, and returns a modified value to cell *4*. Next, cell *0* receives a value from cell *2*, and returns a modified value to that cell. Finally, cell *1* sends a value to cell *0*, for use in its last multiply-add step. The FFT computation is complete after this multiply-add step.

The actions of cell *0* in the FFT algorithm are anomalous in the sense that no other cell performs a multiply-add step during the simulation of every row of the FFT network. In fact, only half of the cells participate in each stage of the computation. In the first row of the *16*-element example of Figure 4-10, the participating cells all have four-bit binary indices of the form $(0cba)_2$. That is, only the first eight cells perform the first multiply-add step, and the outputs of the other

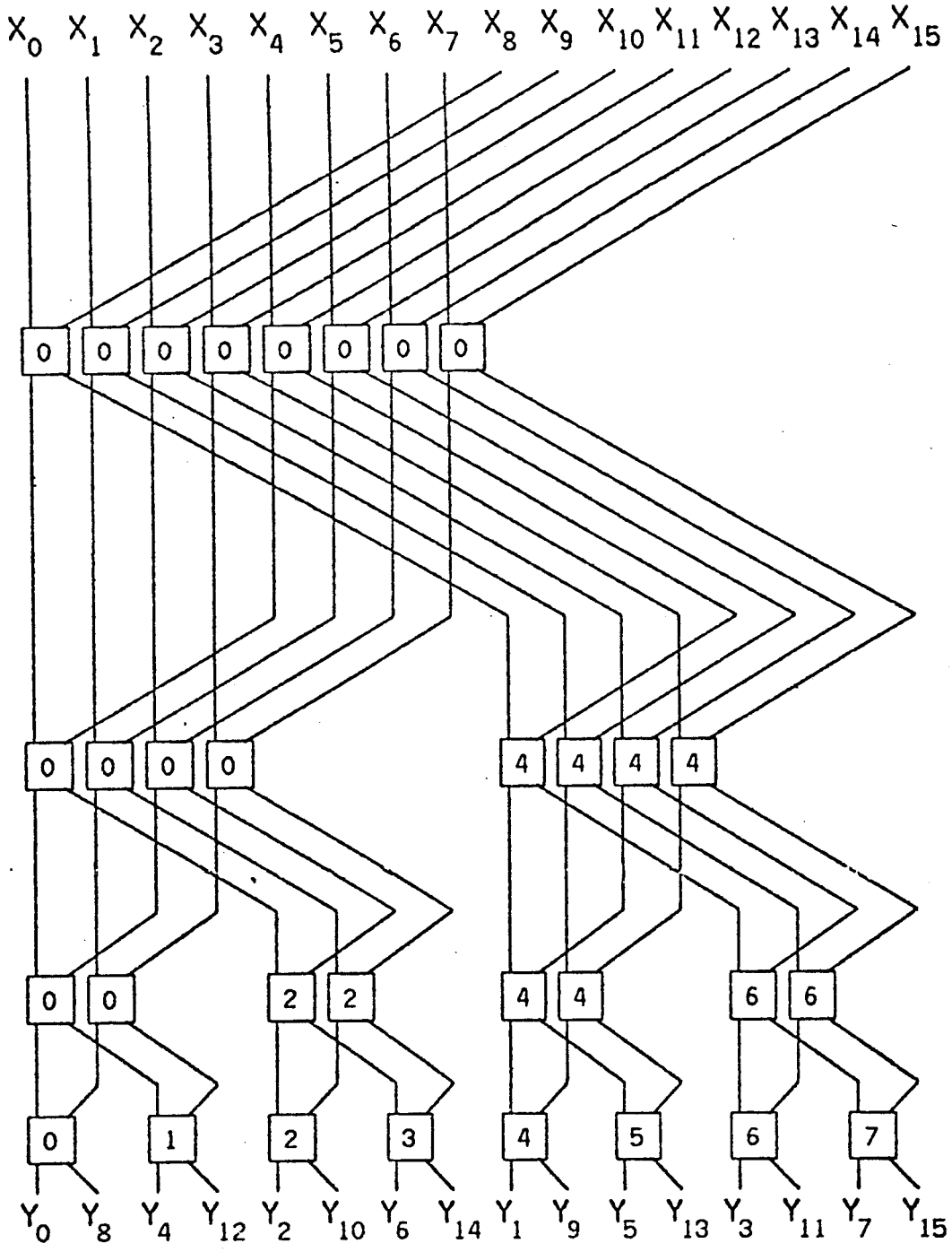

Figure 4-10: A 16-element FFT network.

eight cells will be ignored. Moving to the second row of the FFT network, only cells with binary indices of the form $(d0ba)_2$ do a multiply-add step. In the third and fourth row, cells with indices $(dc0a)_2$ and $(dcb0)_2$ participate in the multiply-add step, respectively.

The actions of cell *0* are also anomalous because it does not have to act as an intermediary in the routing of data values from one cell to another. Most routings on the mesh do involve intermediate cells. For example, the top row of the FFT network of Figure 4-10 involves a distance-*8* routing: the data from cell *8* reaches cell *0* by way of cell *4*, passing over two vertical interconnections. Note that all distance-*8* routings can be performed (in parallel) by way of one intermediate cell at distance *4* from the sender and the receiver.

On an $N = n^2$ cell mesh, both distance *1* and distance n routings are called *unit distance* routes because they can be accomplished over horizontal or vertical connections with no intermediate cells. A unit distance route takes time t_R.

The total time taken by the data movement during an FFT can be evaluated in terms of the number of unit distance routes performed. In the case of the *16*-element FFT of Figure 4-10, the top row's distance-*8* routing takes two unit distance routes $(2t_R)$. After the first multiply-add step, there is another distance-*8* routing followed by a distance-*4* routing, taking time $3t_R$. After the second multiply-add step, there is a distance-*4* routing and a distance-*2* routing, which also takes time $3t_R$. The third multiply-add step is followed by a distance-*2* and a distance-*1* routing, for another $3t_R$ units of time. The fourth multiply-add step is followed by a (superfluous) distance-*1* routing. The total routing time in the computation of a *16*-element FFT is thus $12t_R$.

In the general case of an N-cell mesh performing an N-element FFT, cell indices can be represented as *(log N)*-bit binary numbers. There are *log N* rows in the FFT network, the *kth* of which is simulated by a multiply-add step in cells with a zero bit in the *((log N) − k)th* bit of their index. A distance-$(N/2^k)$ routing is performed before and after this multiply-add step. A summation of the time contributions of all rows k of the FFT network will give the total amount of routing time. Similarly, the amount of time taken by the multiply-add computations can also be evaluated in terms of t_M, the time taken by a single multiply-add step. This line of reasoning leads to the following lemma.

Lemma 1: An N-element FFT can be performed on an N-cell square mesh in time $T = O(N^{1/2}t_R + (\log N)t_M)$, if a unit-distance route takes time t_R and a multiply-add step takes time t_M.

Proof. There are $\log N$ rows in an N-element FFT network. The cells of the mesh simulates row k of the network by performing two distance-$(N/2^k)$ routings and one multiply-add step. When $k < (\log N)/2$, the mesh's vertical interconnections should be used to perform the routings in minimal time. Otherwise, the horizontal connections should be used.

The total time for the FFT is thus

$$T = \sum_{1 \le k < (\log N)/2} (2(N/2^k)/N^{1/2} t_R + t_M) + \sum_{(\log N)/2 \le k < \log N} (2(N/2^k)t_R + t_M), \quad (4.3)$$

so that

$$T = 4(N^{1/2} - 1)t_R + (\log N)t_M. \ \square \quad (4.4)$$

This completes the description of the way in which the FFT algorithm can be performed on mesh connections. The next subsection gives a VLSI implementation of a multiply-add cell. A mesh composed of N of these cells can compute an FFT in the manner described above.

4.3.2 The multiply-add cell

Each multiply-add cell in a mesh implementation of the FFT must perform two functions. First, it must be able to perform a multiply-add step on two $(\log M)$-bit numbers X_0 and X_1, computing

$$Y_0 = X_0 + \alpha_j X_1 \quad (4.5)$$

and

$$Y_1 = X_0 - \alpha_j X_1. \quad (4.6)$$

Here, α_j is the power of α used in the jth stage of the FFT computation. Each multiply-add cell uses a different list of α_j values, corresponding to the computations performed in its column of the FFT network. A multiply-add step takes time t_M, in the terminology of Lemma 1.

Second, each cell must be able to send and receive messages (data values) to and from its immediate neighbors in the mesh. Such a transmission is called a unit-distance route and takes time t_R in the terminology of the previous subsection.

By the result of Lemma 1, the time performance of an N-cell square mesh will be affected much more strongly by t_R than by t_M.

This observation suggests that a bit-serial method be used in the multiply-add computation to save area. Also, a bit-parallel routing method should be used to minimize t_R.

Figure 4-11 shows the structure of a multiply-add cell that does bit-serial computation but has bit-parallel computation paths to and from its immediate neighbors. There will be N such cells in a complete mesh, arranged in a square pattern. Each cell is $O(\log N)$ units wide by $O(\log N)$ units long, so that N nodes occupy $O(\log^2 N)$ area. Note that there are $\log M$ wires in each of the parallel communication paths, so that a $(\log M)$-bit word can be transmitted to a neighbor in a single operation. However, it will take $O(\log N)$ time to get a word out of a transceiver and into the arithmetic circuits, since there is only a serial data path between these two elements. (Recall that $\log M = O(\log N)$ by assumption U4 of Chapter 2.)

There are four control wires for the vertical (resp. horizontal) transceiver, telling it to transmit data in parallel upwards (to the right) or downwards (to the left), or else to shift data serially into or out of the arithmetic circuit. The internal structure of a transceiver is shown in Figure 4-12. The circles are logic nodes forming a shift register that can be accessed either serially (shift in, shift out) or in parallel (up, down). The large triangular drivers must be able to send a signal down an $O(\log N)$ unit wire. Hence they have area $O(\log N)$. Since the entire transceiver must fit in an $O(\log N)$ by $O(\log N)$ unit area, these drivers must be oriented vertically, as indicated in Figure 4-12.

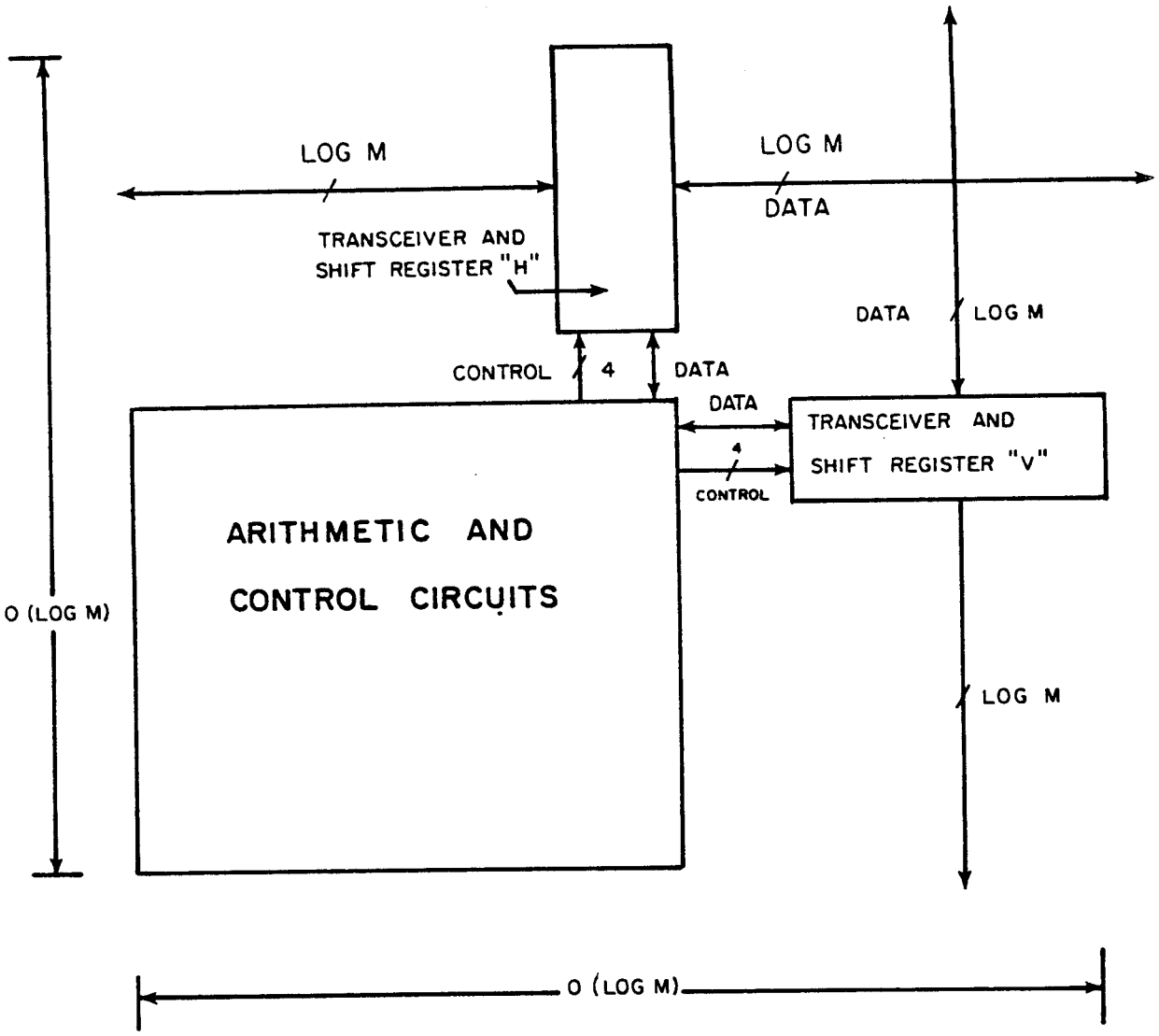

Figure 4-11: Multiply-add cell for the mesh.

Lemma 2: The multiply-add cell of Figures 4-11 and 4-12 has a routing time $t_R = O(\log\log N)$, exclusive of the time it takes to shift data into and out of the transceivers (the shifting time will be included with t_M).

Proof. The delay of a driver of area $O(\log N)$ is $O(\log\log N)$, by assumption U5 of Chapter 2. □

The total routing time for an N-element FFT computation is thus $O(N^{1/2}t_R) = O(N^{1/2}\log\log N)$. This time bound could be improved to $O(N^2)$ if a few more assumptions were added to the upper bound model of computation. As

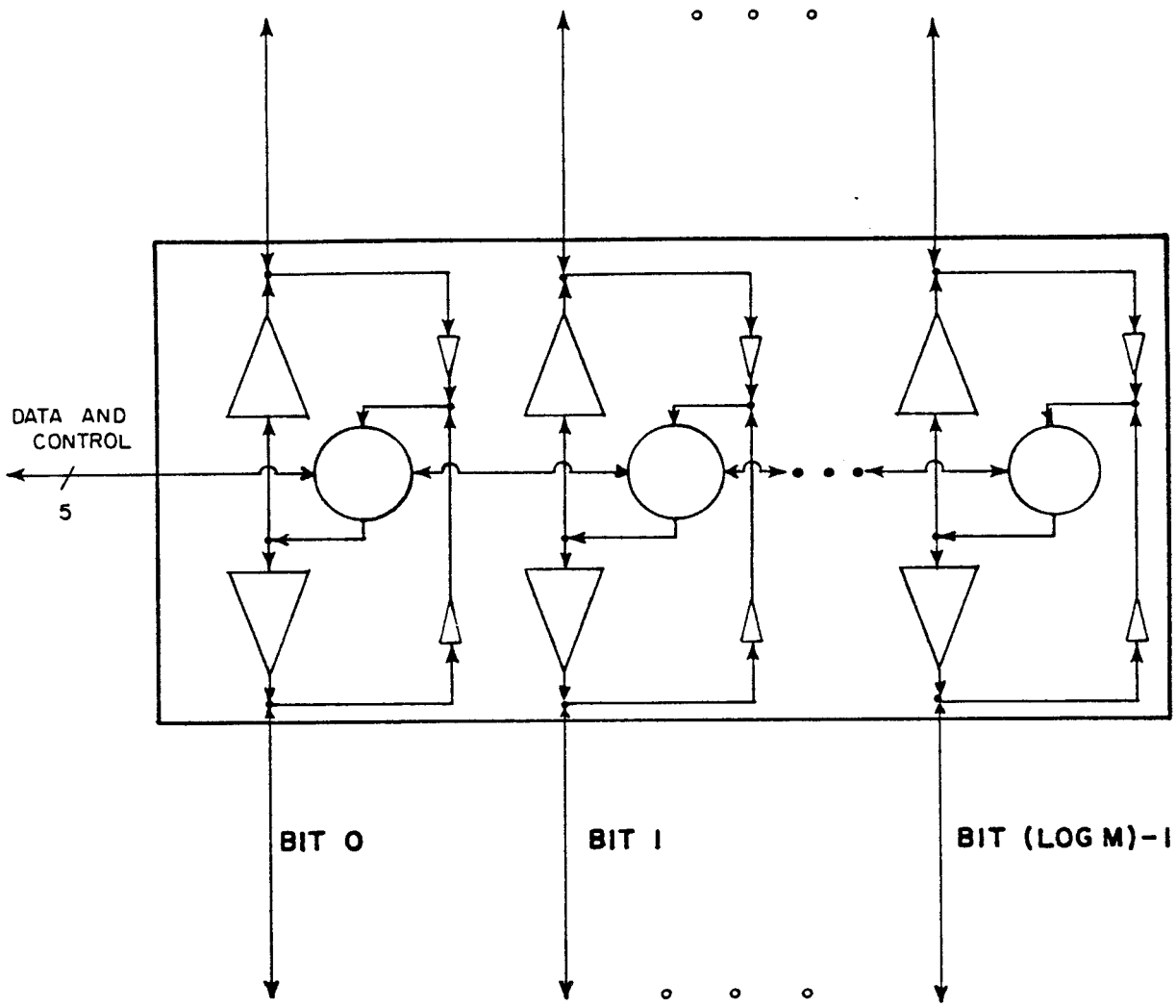

Figure 4-12: Transceiver for the mesh.

noted in the proof of Lemma 2, the factor of *O(loglog N)* arises from the delay of the drivers: each has *O(loglog N)* stages of amplification. Such amplification is necessary beecause the outputs of the arithmetic and control circuitry are obtained from minimal-sized transistors. However, there is no need for much gain in the data paths from one transceiver to the next. This observation suggests the following construction for a transceiver. Its drivers, receivers and shift registers could be built entirely from *O(log N)*-area transistors.[2] The scaled drivers would work with *O(1)*

[2]Even though it is quite feasible to "scale" circuitry in this way, the concept was not included in the upper bound model of computation in order to keep it as simple as possible. Changing the model to allow scaled logic nodes would entail changes in the definition of self-timed regions, since an otherwise-legal logic node might be too large to fit in a self-timed region.

delay, reducing the unit-distance routing time t_R to *O(1)* time units. The multiply-add time t_M would be increased slightly, since there would be *O(loglog N)* stages of amplification between the outputs of the arithmetic and control circuitry and the inputs of the transceivers. However, if these stages were individually clocked, the amplifier would have a throughput of one bit per time unit and only *O(loglog N)* delay. The additional delay would be insignificant in comparison to the *O(log N)* time units required to shift the data in and out of the arithmetic circuitry.

The arithmetic and control circuits of the multiply-add cell are shown in Figure 4-13. This circuitry can be thought of as a microcoded processor having a control program of *O(log N)* instructions that are each *O(log N)* bits wide. Each instruction has three fields: a timer word to indicate the number of clock cycles that the current instruction should persist, an α_j value to be used by the arithmetic unit, and a control field for the transceivers and the multiplexers in the upper right hand corner. As will be seen below, no instruction will persist for more than $O(N^{1/2} \log\log N)$ units of time, so that an *O(log N)*-bit timing word will be sufficiently long.

The arithmetic unit has three serial data inputs and two serial outputs labelled α_j, X_0, X_1, Y_0 and Y_1. It implements equations (4.5) and (4.6), producing Y_0 and Y_1 from X_0 and $\alpha_j X_1$ in a bit-serial fashion.

The Appendix gives a control program for cell *0* in an *N*-cell mesh computing an *N*-element FFT. Initially, input X_i to the FFT is contained in the shift register *"V"* of cell *i*'s vertical transceiver. After *log N* stages of computation, output Y_i should be in the horizontal transceiver *"H"* of cell *i*. The comments in the right hand margin of the control program indicate which instructions differ in the programs for cells other than cell *0*. The program for cell *0* is unique in two ways: it participates in each stage of the computation by producing valid Y_0 and Y_1 values from valid X_0 and X_1 values, and it uses the same value of α_j in each stage of the computation (only half of the cells participate in each stage of the computation, and many use several different α_j values).

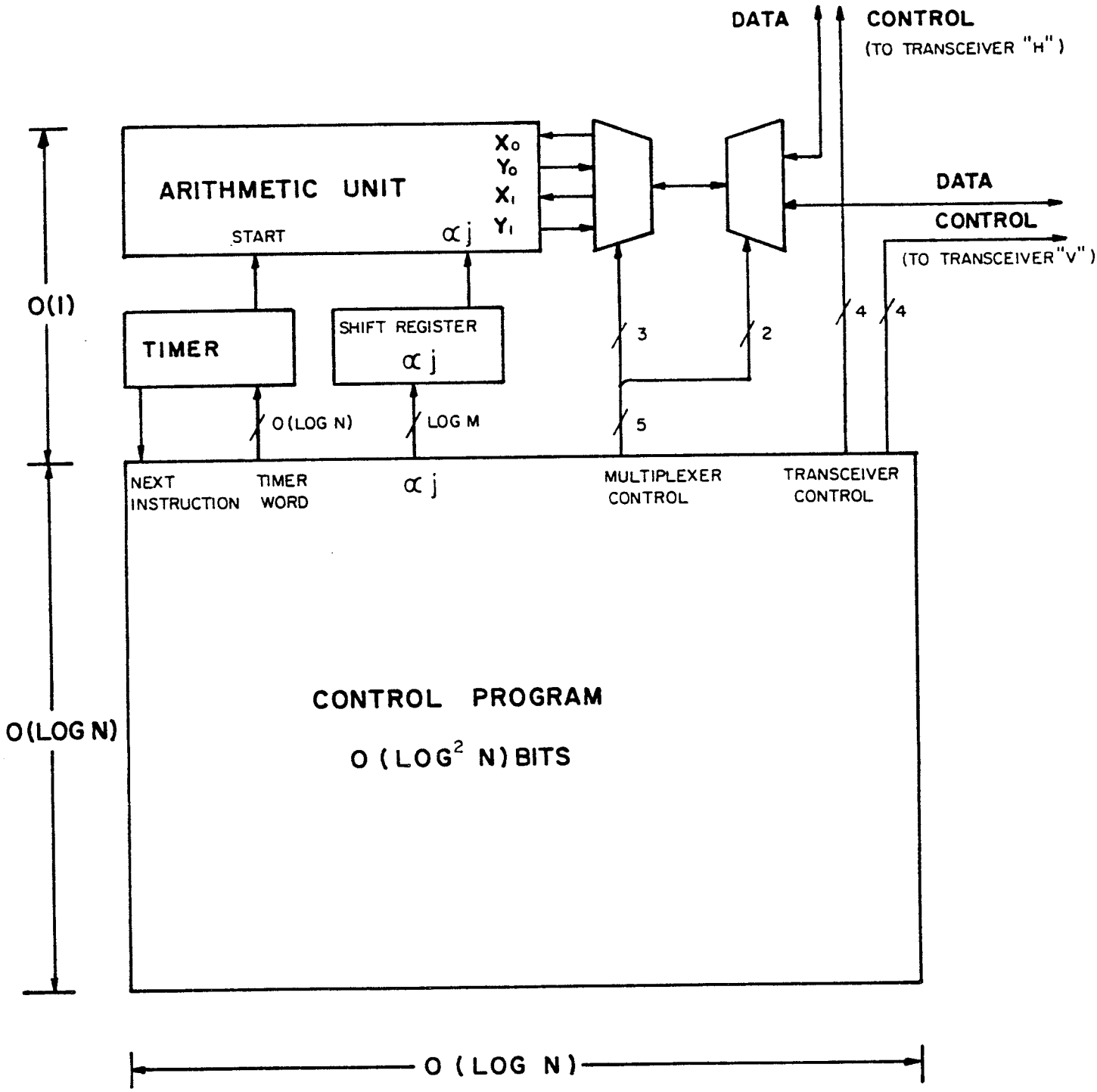

Figure 4-13: Arithmetic and control circuits.

As intimated in the control program for cell *0*, the actual computation of the multiply-add step takes $O(\log^2 M) = O(\log^2 N)$ time. Figure 4-14 shows a small arithmetic unit with this performance, one that fits into an $O(1)$ by $O(\log N)$ unit area.

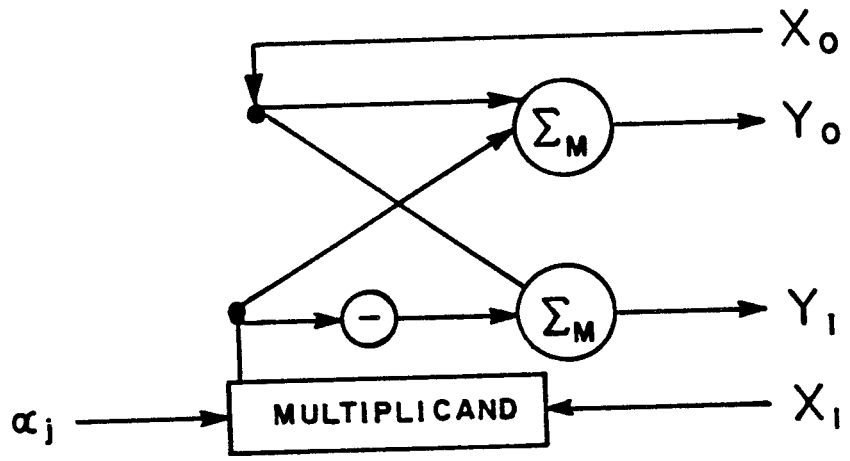

Figure 4-14: An arithmetic unit for an FFT circuit.

The multiplier box of figure 4-14 forms the *mod M* product $\alpha_j x_l$ from the bit-serial inputs x_l and α_j. Its bit-serial output is used in two places. Along one path, it is negated then added *mod M* to x_0 to form the bit-serial output y_l. The other path leads to another bit-serial adder that produces y_0.

The most complicated portion of this construction is the *mod M* multiplier. It consists of $2\lceil \log M \rceil$ carry-save adders that form the product $\alpha_j x_l$. This $2\lceil \log M \rceil$-bit result is then multiplied by $1/M$, with sufficient precision to maintain $\lceil \log M \rceil$ bits of accuracy to the right of the radix point. The bits to the left of the radix point are discarded, and the ones to the right are multiplied by M to form the $\lceil \log M \rceil$-bit result $\alpha_j x_l \bmod M$.

The multiplier thus has three sets of carry-save adders and storage for two constants, $1/M$ and M. The first constant can be expressed as a $2\lceil \log M \rceil$-bit scaled integer, to obtain the necessary accuracy for the second multiplication step. The

other constant is, of course, a $\lceil \log M \rceil$-bit integer. The total amount of hardware in the multiplier is $O(\log M) = O(\log N)$, since a cell of a carry-save adder can be built from $O(1)$ gates. Total time for the three multiplication steps is $O(\log N)$, since each involves only $O(\log M) = O(\log N)$ bits.

A more realistic construction for the multiplier would use only two multiplication steps. The constants α_j and $1/M$ could be multiplied in advance and stored as a single $2\lceil \log M \rceil$-bit number. The $mod\, M$ multiplication of α_j by x_1 could then be performed by multiplying x_1 by α_j/M, then multiplying the fractional part of the result by M.

It is very easy to design the $mod\, M$ adder and negater. Addition can be performed with a bit-serial adder whose output is compared with M in a bit-serial comparator. If the sum is greater than M, then M can be subtracted from it in a bit-serial subtracter. The entire process takes only $O(\log M) = O(\log N)$ time and $O(\log N)$ hadware. The negater is even simpler. Its input is subtracted from M in a bit-serial subtracter. Once again, $O(\log N)$ time and area is sufficient. (This negater produces an erroneous result if its input is zero; however the "excess M" will be removed by the adder connected to its output in Figure 4-14.)

A more realistic circuit for the addition portion of the arithmetic unit would combine the negation step with the following addition step, saving $O(\log N)$ time and area.

The overall performance of the multiply-add cell is described by the lemma below, which summarizes the preceding discussion.

Lemma 3: The multiply-add cell of Figures 4-11 through 4-14 has a multiply-add time $t_M = O(\log N)$ and area $A = O(\log N)$.

We are now in a position to prove the following theorem about the performance of the complete mesh-based FFT circuit.

Theorem 4: An N-element FFT can be performed in $O(N \log N)$ area and $O(N^{1/2} \log\log N)$ time on N multiply-add cells arranged in a square mesh.

Proof. By Lemma 1, an N-element FFT can be performed in time $T = O(N^{1/2})t_R + (\log N)t_M)$ on an N-cell mesh. The unit routing time for this structure, t_R, is $O(\log\log N)$ by Lemma 4-12. Lemma 3 gives $t_M = O(\log N)$. Thus $T = O(N^{1/2} \log\log N)$.

The total area is $O(N \log^2 N)$ since there are N cells of $O(\log^2 N)$ area each. □

The following corollaries state the combined area-time performance of the mesh-based FFT circuit.

Corollary 5: $AT^2 = O(N^2 \log^2 N \log\log^2 N)$ for the N-element FFT.

Corollary 6: $AT^{2x} = O(N^{1+x} \log^2 N \log\log^{2x} N)$ for the N-element FFT.

Note that the FFT circuit of this subsection is very nearly optimal under the AT^2 metric developed in Chapter 3, for it is at most $O(\log\log^2 N)$ from the optimal $AT^2 = \Omega(N^2 \log^2 N)$ of Theorem 11. It is also nearly optimal under the AT^{2x} metric, since it is $O(\log\log^{2x} N)$ from the optimal value of AT^{2x}.

Also note that the total multiply-add time of $(\log N)t_M$ is completely dominated by the routing time $N^{1/2}t_R$. There is thus no incentive to improve the multiply-add circuitry, for such improvements will not affect the asymptotic performance of the complete FFT circuit.

The next subsection shows how an FFT can be computed using a circuit built around a shuffle-exchange pattern recirculation network.

4.3.3 The FFT on shuffle-exchange connections

The shuffle-exchange pattern is a natural choice for a recirculation network solving the FFT [Stone 71]. This subsection revises Stone's FFT circuitry to apply it

to the VLSI model of computation. In contrast to the mesh-based construction of the previous section, only one routing step need be made between multiply-add steps. Circuits based on the shuffle-exchange pattern are thus faster than mesh-based ones, since the performance of the mesh is limited by its routing time. However, the area-time tradeoff results of Chapter 3 imply that shuffle-exchange circuits must be much larger than mesh-based ones, to counterbalance their improved time performance.

The shuffle-exchange graph is defined on $N = 2^N$ nodes numbered from *0* to $N-1$. Each node has two incoming and two outgoing edges. The outgoing edges for node i connect to node $(2i+\lfloor 2i/N \rfloor) \bmod N$ (a *shuffle* connection), and to node $i \oplus 1$ (an *exchange* connection). Here $\oplus$ represents the bitwise exclusive-or operation which, in this case, complements the least significant bit of i. Examples of shuffle-exchange graphs have appeared earlier in this thesis, in Figures 2-3 and 3-4. An *8*-node shuffle-exchange graph is drawn below in Figure 4-15.

The following subsection treats the problem of embedding the shuffle-exchange graph in the plane, according to the embedding rules of Chapter 3. The current subsection examines the way in which an FFT can be performed on $N/2$ cells connected in a shuffle-exchange pattern. (Note that this subsection's construction solves an N-element FFT on $N/2$ cells, not on N cells as in the mesh-based construction.)

The FFT algorithm was defined in Section 4.1 as a network of $\frac{1}{2}N(\log N)$ cells, each performing one multiply-add step. As we saw in Subsection 4.3.1, the recursive construction of the FFT network lends itself to a mesh-based implementation in which each of N cells did the work of all the cells in one of the N columns. As it stands, the FFT network is *not* suited for implementation on a shuffle-exchange pattern, for there is a different pattern of data movement between each row of the network.

However, the cells in each row of an FFT network can be rearranged so that the interconnections between each row form the same pattern, a "perfect shuffle" [Pease

68, Stone 71]. Figure 4-16 illustrates this rearrangement of Figure 4-10, the *8*-element FFT network. Note that all cells lie in *N/2* columns, and that a multiply-add step occurs in each row of each column.

The linear representation of the shuffle-exchange graph (Figure 4-15) bears the same relationship to the rearranged FFT graph of Figure 4-16 as the linear representation of the mesh (Figure 4-9) does to the FFT graph of Figure 4-10. Each row of the FFT graph corresponds to one stage in the computation on the linear network, that is, to one multiply-add step and one or more passes through the recirculation network.

The rearranged FFT network is ideal for implementation on *N/2* multiply-add cells connected in a shuffle-exchange fashion. Each cell corresponds to two nodes in the shuffle-exchange graph. Cell *i* has all the connections of nodes *2i* and *2i+1* in the *N*-node shuffle-exchange graph (see Figure 4-15). The "exchange" connections of the two nodes are thus internal to the cell, but two "shuffle" connections emanate from the cell. For example, one of the outputs of cell *2* in a *4*-cell shuffle-exchange pattern *(N = 8)* goes to cell *0*, while the other output goes to cell *1*.

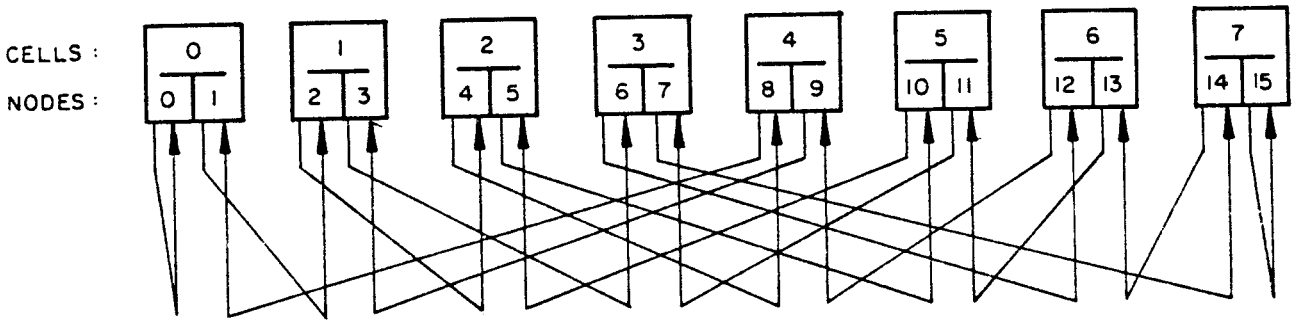

Figure 4-15: A *16*-node shuffle-exchange graph, laid out in a linear fashion. Exchange connections are internal to the cells.

The computation of an *N*-element FFT on an *N/2*-cell shuffle-exchange recirculation network can be described explicitly with reference to Figures 4-15 and 4-16. Initially, each cell contains two of the *N* inputs: cell *i* contains inputs X_{2i} and X_{2i+1} in its registers X_0 and X_1. During each of the *log N* stages of computation,

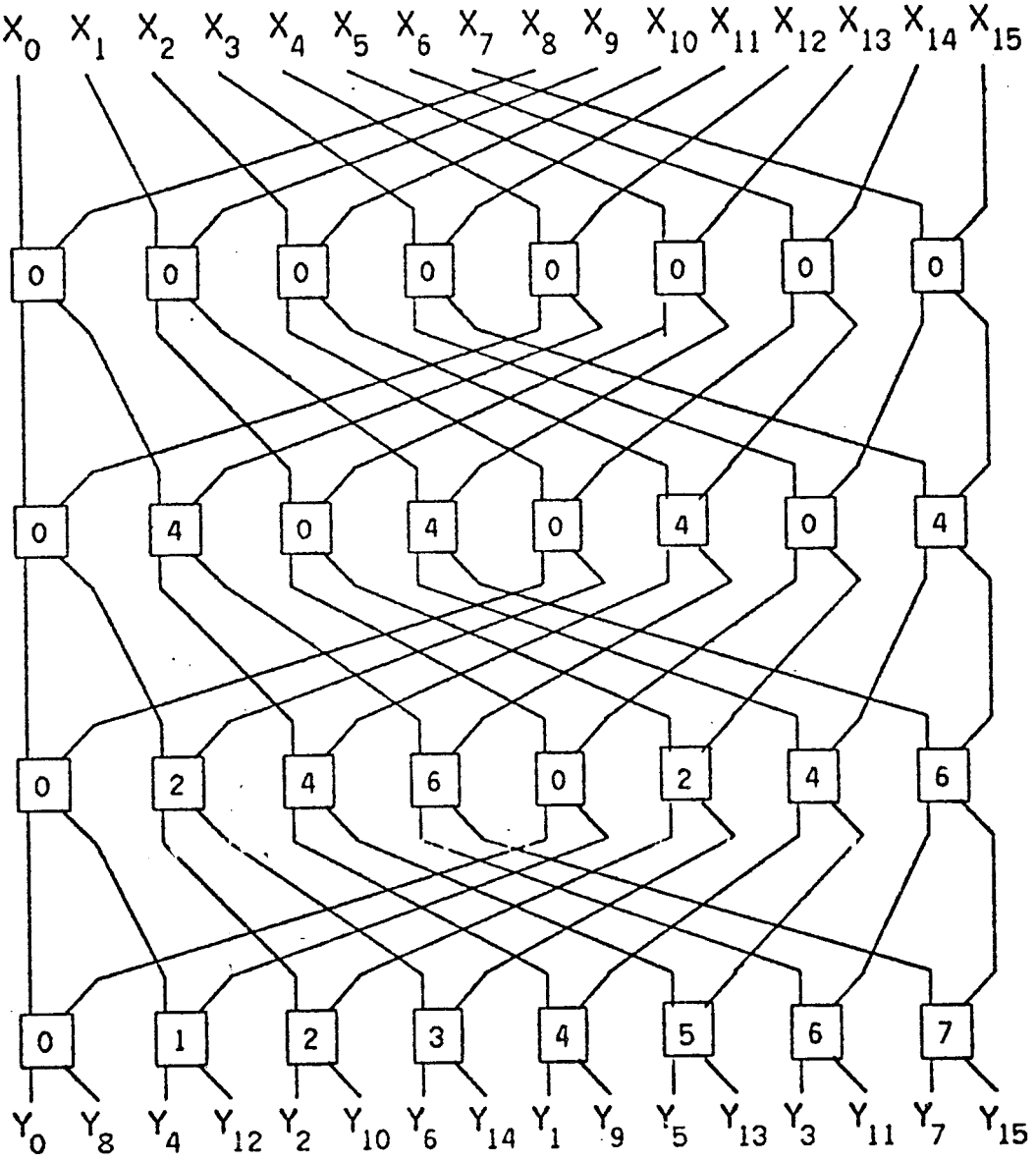

Figure 4-16: A rearranged *16*-element FFT network.

each cell performs a multiply-add step on X_0 and X_1, sends the results Y_0 and Y_1 out over its "shuffle" output connections, and receives new values X_0 and X_1 from its "shuffle" inputs.

The following lemma summarizes the behavior of the shuffle-exchange-based FFT algorithm.

Lemma 7: An N-element FFT can be performed on an $N/2$-cell shuffle-exchange network in time $T = O((\log N)t_R + (\log N)t_M)$, if a routing step takes time t_R and a multiply-add step takes time t_M.

A multiply-add cell for the shuffle-exchange network can be built along the lines of Subsection 4.3.2's construction. Three modifications are necessary, however.

First, there is no reason to provide parallel data paths between cells. Bit-serial connections take up much less area, and are fast enough to keep the multiply-add circuitry busy almost all the time. This was not the case in the mesh construction: multiply-add time was completely dominated by routing time.

A second modification to the multiply-add cell changes its aspect ratio from $O(\log N)$ by $O(\log N)$ to $O(1)$ by $O(\log^2 N)$. The reason for this modification will become apparent in the next subsection, when the shuffle-exchange connections are embedded in the plane.

Finally, the drivers for the Y_0 and Y_1 outputs must be able to handle wires that are $O(N/\log N)$ units long (this is the length of the shuffle connections in the embedding developed in the next subsection).

These three modifications lead to the multiply-add cell shown in Figure 4-17. Note that the control program must be stored in an $O(1)$ by $O(\log^2 N)$ bit array, to fit in the required aspect ratio. This means that $O(\log N)$ time must elapse between instructions (see the Appendix) so that a new instruction can be shifted in.

The following lemma describes the time performance of the shuffle-exchange-based FFT circuit.

Lemma 8: An N-element FFT can be performed on $N/2$ cells interconnected in a shuffle-exchange recirculation pattern in time $T = O(\log^2 N)$.

Proof. A total of $\log N$ routing steps and $\log N$ multiply-add steps are necessary. The time taken by a single multiply-add step is the same as it was for the mesh-based construction, $t_M = O(\log^2 N)$. Total multiply-add time is thus $O(\log^2 N)$.

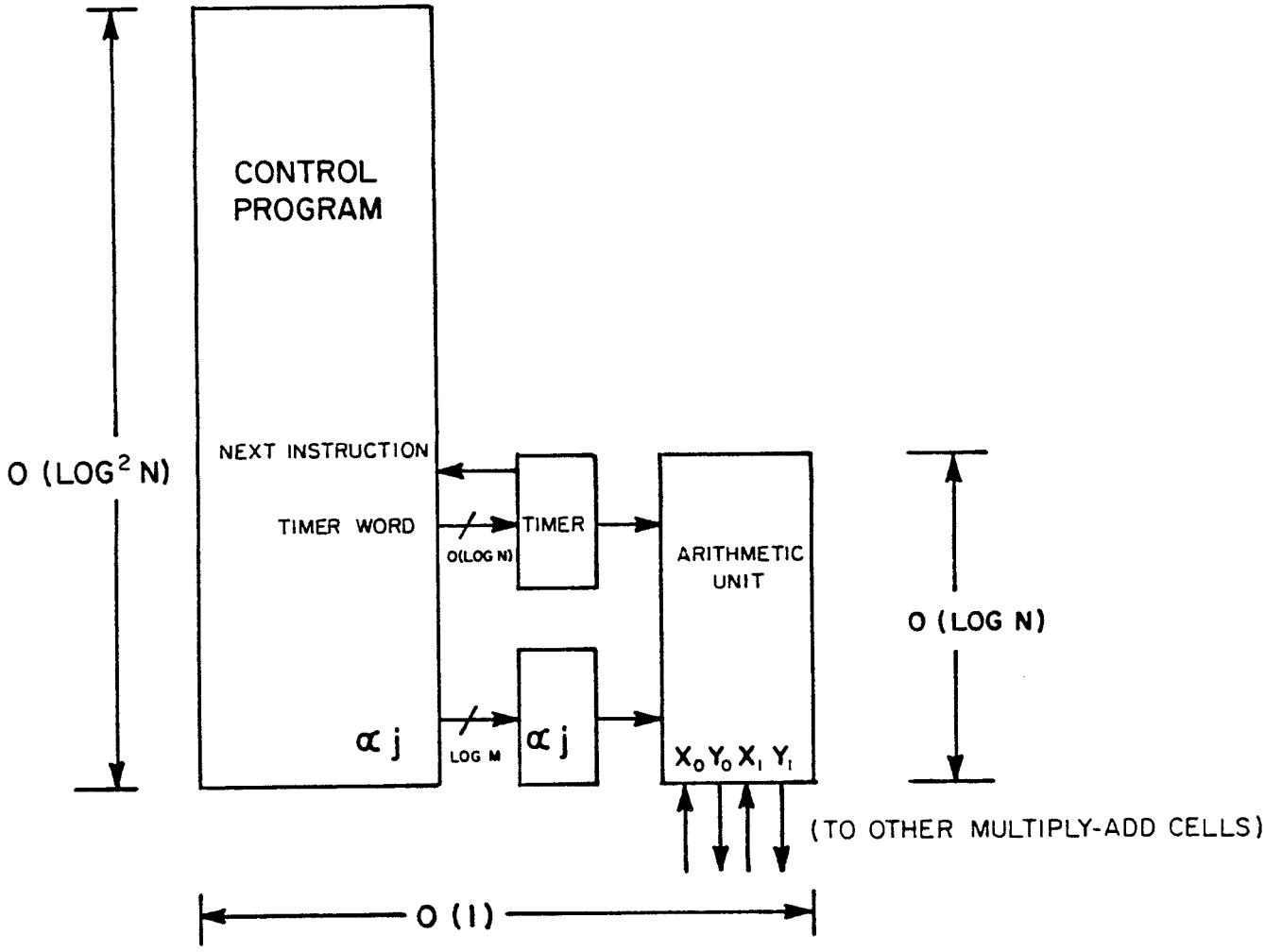

Figure 4-17: Multiply-add cell for a shuffle-exchange network.

Total routing time is also $O(\log^2 N)$, since there are $O(\log N)$ routing steps each involving $\log M = O(\log N)$ bit-serial data transfer operations. These transfers occur at unit bandwidth, so that the transmission of one word takes $O(\log N)$ time. Additionally, there is an $O(\log N)$ delay in the drivers, for the wires between cells are $O(N/\log N)$ units long. (The delay of a driver-wire-reciever circuit is the logarithm of the length of the wire, by assumption U5.)

Note that the time performance of this circuit is optimal: faster multipliers and/or faster instruction loading would not help. The same time performance would be observed if each of the N cells merely sent $O(\log N)$ bits of data to another cell, and repeated the process $O(\log N)$ times. This observation, coupled with

Chapter 3's area-time tradeoff for the DFT, leads to the first theorem in the following section.

4.3.4 Area bounds for the shuffle-exchange graph

This subsection presents upper and lower bounds on the area of a shuffle-exchange graph, when it is embedded in the plane according to the rules of Chapter 2.

Theorem 9: At least $\Omega(N^2/\log^2 N)$ units of area are required to embed a shuffle-exchange graph in the plane under Assumptions L1 through L8.

Proof. Under the *lower* bound model of computation, a communication graph with N unit area nodes could compute an N-element FFT in time $T = O(\log^2 N)$, if the nodes are connected in the shuffle-exchange pattern. Each node could do a multiply-add step in $O(\log N)$ time, the time it takes to shift a word into or out of a node in bit-serial fashion).

Since any communication graph computing the FFT must satisfy $AT^2 = \Omega(N^2 \log^2 N)$ by Theorem 11, this particular graph must have area $A = \Omega(N^2/\log^2 N)$. □.

Corollary 10: The average length of the edges in a planar embedding of the shuffle-exchange graph is $\Omega(N/\log^2 N)$.

Proof. Each node of a shuffle-exchange graph has degree four, so there are $O(N)$ edges in an N-node graph. The nodes themselves take up only $O(N)$ area, so the total area of $\Omega(N^2/\log^2 N)$ must be due to the edges alone. □

The following theorem is a constructive upper bound for embedding the shuffle-exchange graph in the plane. It is optimal to within a factor of $O(\log^{3/2} N)$, judging from the result of the preceding theorem. Closing the remaining gap between the upper and lower bounds is an open research problem. [3]

[3] Recently, Dan Hoey and Charles Leiserson of C-MU [Hoey 80] have produced an $O(N^2/\log N)$ embedding of the shuffle-exchange graph, narrowing the gap between the upper and lower area bounds to $O(\log N)$.

Theorem 11: A shuffle-exchange recirculation network for $N/2$ cells can be laid out in $O(N^2/\log^{1/2} N)$ area, if each cell occupies an $O(1)$ by $O(\log^2 N)$ rectangle.

Proof: preliminary remarks. As described in the circuit construction of Subsection 4.3.3, cell number i corresponds to two nodes in an N-node shuffle-exchange graph, nodes $2i$ and $2i+1$. Placing two cells into one node in this way has the advantage that exchange connections are local to the nodes. The remainder of the proof deals with the problem of arranging the nodes to minimize the length of the shuffle connections.

If *node* indices are expressed as binary numbers of $\log N$ bits each, a shuffle connection exists between nodes i and $(2i+\lfloor 2i/N \rfloor) \bmod N$. Note that this last functional form corresponds to an end-around left shift. Shuffle connections thus connect nodes with the same number of '*1*' bits in the binary representation of their indices.

This observation suggests a partition of the $N/2$ cells into $\log N$ equivalence classes, or neighborhoods. Cell i and cell j are in the same equivalence class iff i and j have the same number of '*1*' bits in their binary representation. Let B_k denote the set of cells with k '*1*' bits.

It is easy to verify that shuffle connections link cells in B_k only with other cells in B_k and with cells in B_{k-1} and B_{k+1}. Consider the even-numbered node in cell i of class B_k, that is, node $2i$. Its shuffle connection must be to another node with exactly k '*1*' bits in its index. The only nodes of that type are the even-numbered nodes in class B_k and the odd-numbered nodes in class B_{k-1}. A symmetrical argument shows that the odd-numbered nodes in B_k connect to odd-numbered nodes in B_k and to even-numbered nodes in B_{k+1}.

Clustering cells by their neighborhoods helps limit the length of the wires implementing the shuffle connections. Unfortunately, the neighborhoods are rather large, for there are only $\log N$ neighborhoods and $N/2$ cells.

The construction. Place the cells in one long horizontal line, grouping them by neighborhood. Orient the cells so that the connections to even-numbered nodes point upwards if they are in even-numbered equivalence classes, downwards otherwise. As indicated in Figure 4-17, the shuffle connections have been localized into $(\log N)+1$ disjoint regions, one for each gap in the sequence $(B_0, B_1, \ldots, B_{(\log N)-1})$.

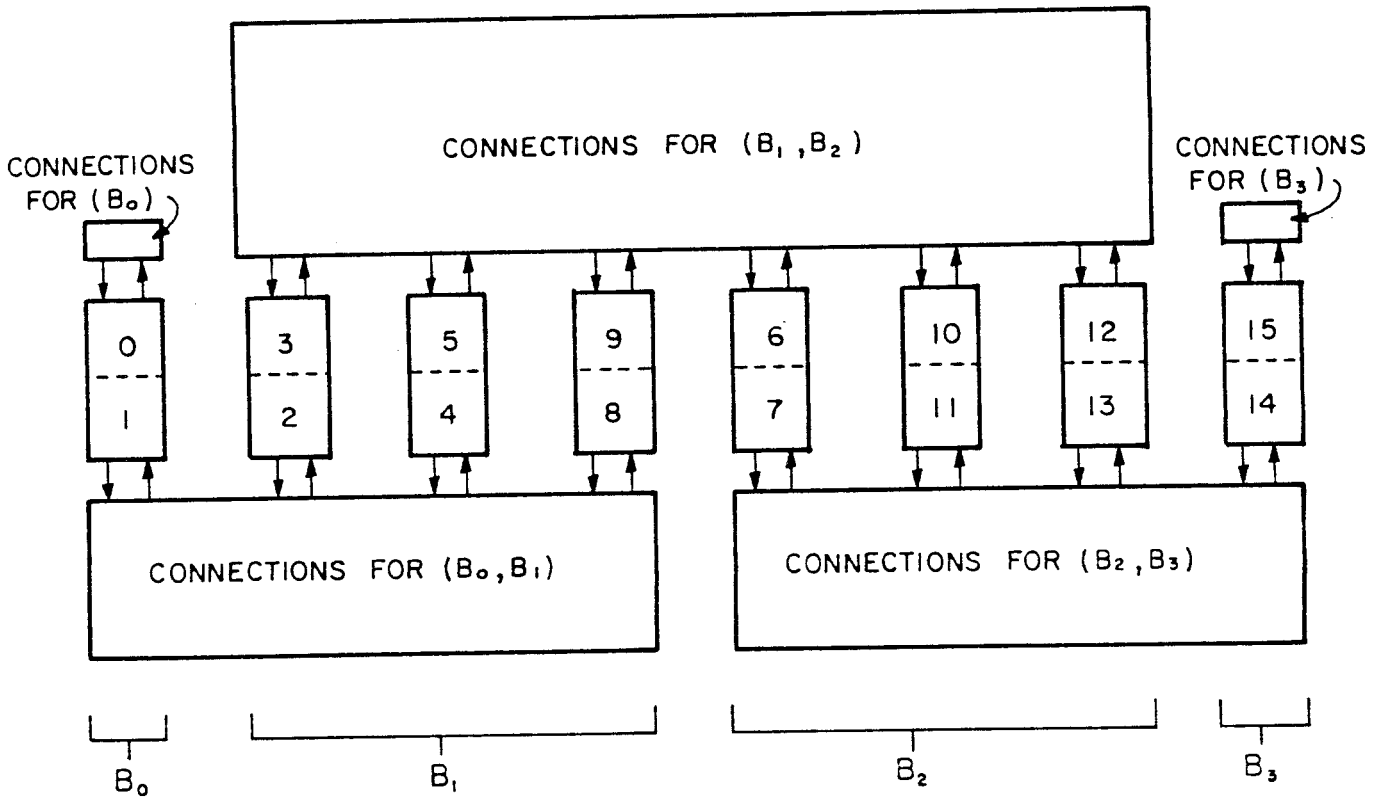

Figure 4-18: Embedding of the shuffle-exchange connections $(N = 16)$.

The shuffle connections for (B_k, B_{k+1}) can be laid out in a rectangular region $O(|B_k| + |B_{k+1}|)$ wide and $O(|B_k| + |B_{k+1}|)$ deep. The width is due to the $O(1)$ width of the $|B_k| + |B_{k+1}|$ cells involved in the shuffle connections of this neighborhood. The depth comes from the $|B_k| + |B_{k+1}|$ wires implementing the connections. Each wire is assigned one unit of depth, so that it can run an appropriate horizontal distance without interference.

The number of cells in B_k is easily counted. It is just the number of cell indices

that have k '1' bits in their binary representation. There are $N/2$ cells, and $(\log N)-1$ bits in each cell index, so that

$$B_k = \binom{\log N - 1}{k}. \tag{4.7}$$

The largest B_k is thus $B_{\lfloor(\log N)/2\rfloor}$, which has size $O(N/\log^{1/2}N)$. The largest neighborhood is $(B_{\lfloor(\log N)/2\rfloor}, B_{\lfloor(\log N)/2\rfloor}+1)$, which may contain wires as long as $O(N/\log^{1/2}N)$.

The entire layout fits in a rectangular region $O(N)$ wide and $O(N/\log^{1/2}N)$ deep. The width of the layout is due to the cell width. The depth is mostly due to the size of the largest neighborhood, since cell depth is asymptotically negligible by comparison. □

Theorem 12: An N-element FFT can be performed in $A = O(N^2/\log^{1/2}N)$ and $T = O(\log^2 N)$ using a circuit based on the shuffle-exchange recirculation pattern.

Proof. Immediate from Lemmas 8 and 11. □

The following corollaries indicate that the shuffle-exchange-based FFT circuit is nearly nearly optimal under the AT^2 metric, for it is only $O(\log^{3/2}N)$ from the optimal performance of $AT^2 = \Omega(N^2\log^2 N)$. However, this same circuit is far from optimal under the AT^{2x} metric for $x < 1$, since it is $O(N^{1-x}\log^{2x-1/2}N)$ from the optimal $AT^{2x} = \Omega(N^{1+x}\log^{2x}N)$.[4]

Corollary 13: $AT^2 = O(N^2\log^{7/2}N)$ for the N-element FFT on a shuffle-exchange-based circuit of Theorem 12.

Corollary 14: $AT^{2x} = O(N^2\log^{4x-1/2}N)$ for the N-element FFT on a shuffle-exchange-based circuit of Theorem 12.

The discussion of Fourier transformation methods is now complete. The next section deals with sorting on the mesh and the shuffle-exchange connections.

---

[4]Recently, Preparata and Vuillemin [Preparata 79] have devised a layout that outperforms the shuffle-exchange-based FFT design described above. Their layout has an area of $O(N^2/\log^2 N)$ and would take $O(\log^2 N)$ time to compute a Fourier transform, if the multiply-add cell of this thesis were used in its implementation. Their construction is thus within a constant factor of the lower bound on AT^2 cost.

4.4 VLSI implementations of sorting

Before reading this section, the reader is advised to review the description of the bitonic sorting network contained in the latter part of Section 4.1.

Subsection 4.4.1 describes the bitonic sorting algorithm as it applies to a VLSI circuit with $N/2$ comparison-exchange cells connected by a shuffle-exchange recirculation network. The construction of the comparison-exchange cell is covered in Subsection 4.4.2. Finally, Subsection 4.4.3 gives a VLSI implementation of the bitonic sort on N cells in a mesh pattern.

4.4.1 Sorting on shuffle-exchange connections

The shuffle-exchange connections are nearly as well-suited for sorting as they are for Fourier transformation. This should not be surprising, for the N-element bitonic sorting network is very similar to the N-element FFT network.

The same approach used to perform an FFT on $N/2$ cells with shuffle-exchange interconnections can be used to perform a bitonic sort, as shown by Stone [Stone 71]. This subsection adapts his circuits to the VLSI model of computation.

The time taken to sort N elements on a shuffle-exchange-based circuit is bounded by the following lemma.

> **Lemma 15:** An N-element sort can be performed on $N/2$ cells interconnected with a shuffle-exchange recirculation network in time $T = O((\log^2 N)t_R + (\log^2 N)t_C)$, if it takes t_R units of time to perform a routing step and t_C units of time to perform a comparison-exchange step.

Proof. Refer to Figures 4-4 through 4-7 for the construction of the bitonic sorting network. The bottom half of Figure 4-6 forms an N-element bitonic merger, which is isomorphic to an N-element FFT network. The reader may verify this by comparing the bottom three rows of Figure 4-5 with Figure 4-2.

Note that successive rows of these networks specify distance $N/2, N/4, \ldots, 1$

routings. By Lemma 8, these bottom rows can be simulated in time $O((\log N)t_R + (\log N)t_C)$, where the comparison time t_C takes the place of the multiply-add time t_M of the FFT algorithm.

The top half of the bitonic sorting network is formed recursively from two half-sized sorting networks. As in the case of the full-sized network, the last $(\log N)-1$ rows of these half-sized networks can be specified by a geometrically decreasing set of routings: distance $N/4, N/8, \ldots, 1$. Note that these two networks are contained side-by-side in the last $(\log N)-1$ rows of an N-element Fourier transform network. Again by analogy with the Fourier transform circuit, a total of $O((\log N)t_R + (\log N)t_C)$ time is sufficient to simulate these rows on a shuffle-exchange network, if a null comparison-exchange is performed after the (seemingly unnecessary) distance $N/2$ routing of the first step of the FFT algorithm. Actually, this routing is not unnecessary, for it does result in a net movement of data among the cells, in preparation for the distance $N/4$ routing.

Continuing with the recursive construction of the bitonic sorting network, the top halves of the two half-sized sorting networks are themselves formed of four quarter-sized sorting networks. The routings of the last $(\log N)-2$ rows of these quarter-sized networks are distance $N/8, N/16, \ldots, 1$. Once again, the FFT algorithm can be used to simulate these routings in $O((\log N)t_R + (\log N)t_C$ time, if *two* null comparison-exchanges are followed by $(\log N)-2$ comparison-exchange steps.

The recursive construction is complete at a depth of $(\log N)-1$. The last network in the construction is a 4-sorter formed of two 2-sorters and one 4-merger. The 2-sorters are identical to 2-mergers, so the entire construction may be visualized as $N/2$ 2-mergers above $N/4$ 4-mergers above $N/8$ 8-mergers above . . . one N-merger. Each of the $(\log N)$ levels of mergers can be simulated in $O((\log N)t_R + (\log N)t_C)$ time, for total time $O((\log^2 N)t_R + (\log^2 N)t_C)$. □

The preceding lemma has perhaps obscured the simplicity of the sorting process on a shuffle-exchange network. A sort consists of $\log^2 N$ passes through the network,

where a (possibly null) comparison-exchange operation intervenes between each pass. The operations required of a cell are thus very simple. During each stage of computation it must do one of three things: a null comparison-exchange,

$$Y_1 \leftarrow X_1$$
$$Y_2 \leftarrow X_2, \tag{4.8}$$

a "$\leftarrow$" comparison-exchange,

$$Y_1 \leftarrow max(X_1, X_2)$$
$$Y_2 \leftarrow min(X_1, X_2), \tag{4.9}$$

or a "$\rightarrow$" comparison-exchange,

$$Y_1 \leftarrow min(X_1, X_2)$$
$$Y_2 \leftarrow max(X_1, X_2). \tag{4.10}$$

The next subsection describes the construction of a comparison-exchange cell that is capable of performing these operations.

4.4.2 The comparison-exchange cell

Figure 4-19 illustrates a construction of a comparison-exchange cell that can fit into either an $O(log\ N)$ by $O(log\ N)$ area or into an $O(1)$ by $O(log^2 N)$ area.

The serial comparator logic at the top of the figure can be built with $O(1)$ gates in a straightforward manner, as long as the inputs are presented most-significant bit first [Moravec 79]. This logic is responsible for putting the larger or smaller of the two inputs X_1 and X_2 onto the output lines Y_1 and Y_2; the "direction" of the comparison-exchange is dictated by the two-bit state vector stored in the shift registers below. For concreteness, the state-vector assignment may be specified as follows:

$$00 \Rightarrow Y_1 \leftarrow X_1, \quad Y_2 \leftarrow X_2, \tag{4.11}$$
$$10 \Rightarrow Y_1 \leftarrow max(X_1, X_2), \quad Y_2 \leftarrow min(X_1, X_2), \text{ and} \tag{4.12}$$
$$11 \Rightarrow Y_1 \leftarrow min(X_1, X_2), \quad Y_2 \leftarrow max(X_1, X_2). \tag{4.13}$$

Since the sorting algorithm is complete after $log^2 N$ passes through a shuffle-

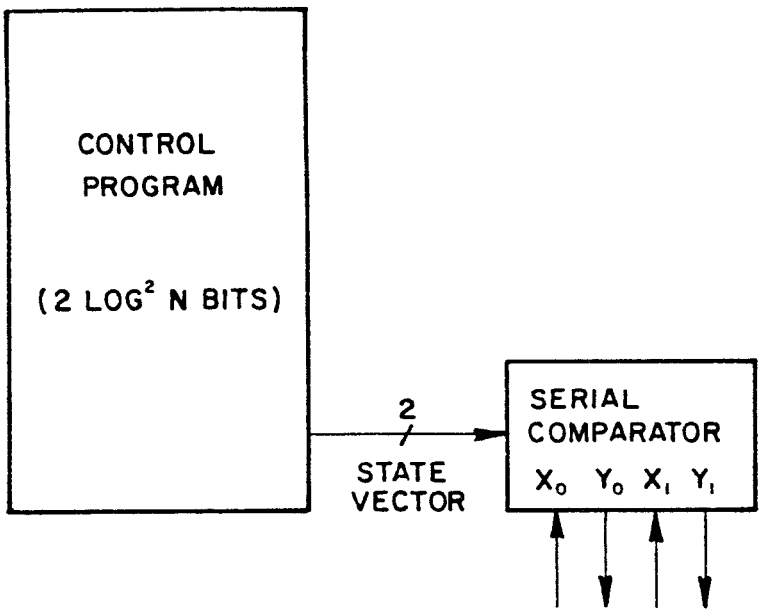

Figure 4-19: The comparison-exchange cell.

exchange based recirculation network, $2log^2N$ bits of state vector suffice to define all operations of the comparison-interchange cell. The shift registers holding the state vector should be clocked once every t_C time units, if t_C is defined (as in the previous subsection) as the time taken for a comparison-exchange step. The value of t_C is $O(log\ N)$, since each bit of the inputs is sent to one of the outputs after $O(1)$ delay. (Note that this performance is possible only if the *most*-significant bits of the inputs are sent first.)

The discussion above is summarized by the following lemma and theorem.

Lemma 16: The comparison-exchange cell of Figure 4-19 has a comparison-exchange time t_C of $O(log\ N)$ and an area of $O(log^2N)$. Its aspect ratio may be either square or $O(1)$ by $O(log^2N)$, depending on how the shift registers are "folded."

Theorem 17: An N-element sort can be performed in $O(N^2/log^{1/2}N)$ area and $O(log^3N)$ time on $N/2$ comparison-exchange cells arranged in a shuffle-exchange pattern.

Proof. By Lemma 15, an N-element sort can be done in time $T = O((log^2N)t_R + (log^2N)t_C)$ on $N/2$ cells connected in a shuffle-exchange

pattern. The unit routing time t_R is the same as it was in the FFT circuit of Lemma 8, since the words still have $O(\log N)$ bits. Thus $t_R = O(\log N)$, and $t_C = O(\log N)$ by the preceding Lemma. Finally, the embedding of Subsection 4.3.4 may be used to give an area bound of $O(N^2/\log^{1/2} N)$. □

The following corollaries are immediate.

Corollary 18: $AT^2 = O(N^2 \log^{11/2} N)$ for an N-element sort performed on $N/2$ comparison-exchange cells connected with a shuffle-exchange recirculation network.

Corollary 19: $AT^{2x} = O(N^2 \log^{6x-1/2} N)$ for an N-element sort performed on $N/2$ comparison-exchange cells connected with a shuffle-exchange recirculation network.

The shuffle-exchange-based sorting circuit is thus nearly optimal under the AT^2 metric. It is only $O(\log^{7/2})$ from the optimal performance of $AT^2 = \Omega(N^2 \log^2 N)$. However, this same circuit is far from optimal under the AT^{2x} metric (unless $x = 1$), for it is $O(N^{1-x} \log^{4x-1/2} N)$ from the optimal $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$.

4.4.3 Sorting on mesh connections

This subsection shows how the bitonic sorting algorithm can be used to sort N elements on N comparison-exchange cells arranged in a mesh. The same approach is used as in the other recirculating constructions of this thesis: each stage of computation on the N cells corresponds to one row of the bitonic sorting network. Since there are $O(\log^2 N)$ rows in the sorting network, a total of $O(\log^2 N)$ comparison-exchange steps must be performed. However, the total number of unit-distance routing steps is variable, depending on the way in which the data is distributed among the cells during the course of the computation.

To minimize the number of routing steps, the cells should be indexed in "shuffled row-major order" [Thompson 77]. In this scheme, a cell is given the n-bit binary index $b_n b_{n/2} b_{n-1} b_{n/2-1} b_{n-2n/2-2} b_{n-3} b_{n/2-3} \cdots b_{n/2+3} b_3 b_{n/2+2} b_2 b_{n/2+1} b_1$ if its natural "row-major" index would be $b_n b_{n-1} b_{n-2} b_{n-3} \cdots b_{n/2+3} b_{n/2+2}$

$b_{n/2+1}\,b_{n/2}\,b_{n/2-1}\,b_{n/2-2}\,b_{n/2-3}\cdots b_3\,b_2\,b_1$ (think of printing each of the bits of a row-major index on a playing card, then "shuffling" the top half of the deck $b_n\,b_{n-1}\,b_{n-2}\,b_{n-3}\cdots b_{n/2+3}\,b_{n/2+2}\,b_{n/2+1}$ with the bottom half $b_{n/2}\,b_{n/2-1}\,b_{n/2-2}\,b_{n/2-3}\cdots b_3\,b_2\,b_1$). Figure 4-20 illustrates the row-major and shuffled row-major indexing of a mesh with $N = 16$ cells. Note that the cell in the lower left corner has row-major index $12_{10} = 1100_2$ and hence shuffled row-major index $10_{10} = 1010_2$.

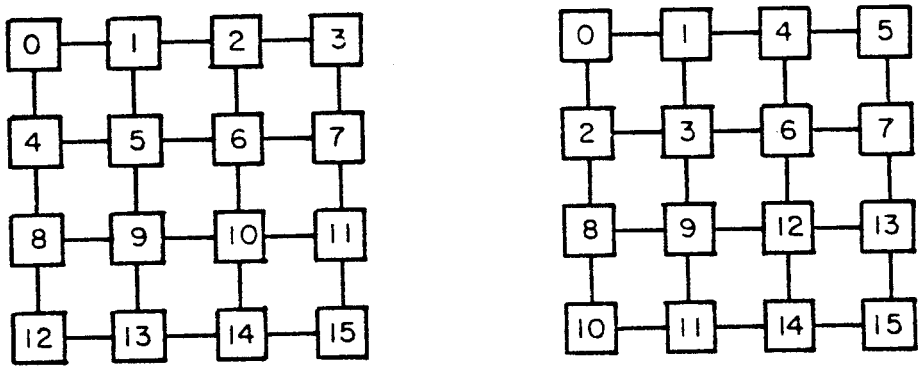

Figure 4-20: Row-major and shuffled row-major indexing schemes for the 16-cell mesh.

The shuffled row-major indexing scheme has two important properties for sorting algorithms. First of all, it is similar to the row-major indexing scheme in that it is simple to move data between pairs of cells whose indices differ only in the *kth* bit. In general, 2^j horizontal or vertical routes will move data between all such pairs of cells. For example, a distance-*2* routing is two horizontal steps in the row-major indexing scheme and one vertical step in the shuffled row-major indexing scheme. A distance-*8* routing is two vertical steps in either scheme. (It should be obvious that any indexing scheme obtained by permuting the bits of the row-major cell indices must preserve this property. However, it is difficult to verify the property by direct examination of the linear form of the shuffled row-major indexing scheme in Figure 4-21.)

A second property of the shuffled row-major indexing scheme distinguishes it from the row-major indexing scheme, a property that leads to its improved time performance on the bitonic sort. The number of routing steps required to do a distance-2^j routing in the shuffled row-major indexing scheme is never more than

the number of steps required for a distance-2^{j+1} routing. (Compare this with the row-major indexing scheme, in which a distance-$N^{1/2}/2$ routing is $N^{1/2}/2$ horizontal steps but a distance-$N^{1/2}$ routing is only one vertical step.) Since there are many more short routings than long routings in the bitonic sorting algorithm, this property of monotonicity makes the shuffled row-major indexing scheme preferable to the row-major indexing scheme. (It turns out that the difference between the two schemes is asymptotically significant. The shuffled row-major indexing scheme is a factor of $O(\log N)$ faster than the row-major indexing scheme.)

A recirculating bitonic sorting algorithm is obtained from the N-element bitonic sorting network and the linear form of the shuffled row-major indexing scheme for the N-cell mesh shown in Figure 4-21. Each row of the network is simulated in turn by the N cells. Connections between rows are simulated by data routings among the cells. The following lemma bounds the time performance of this algorithm.

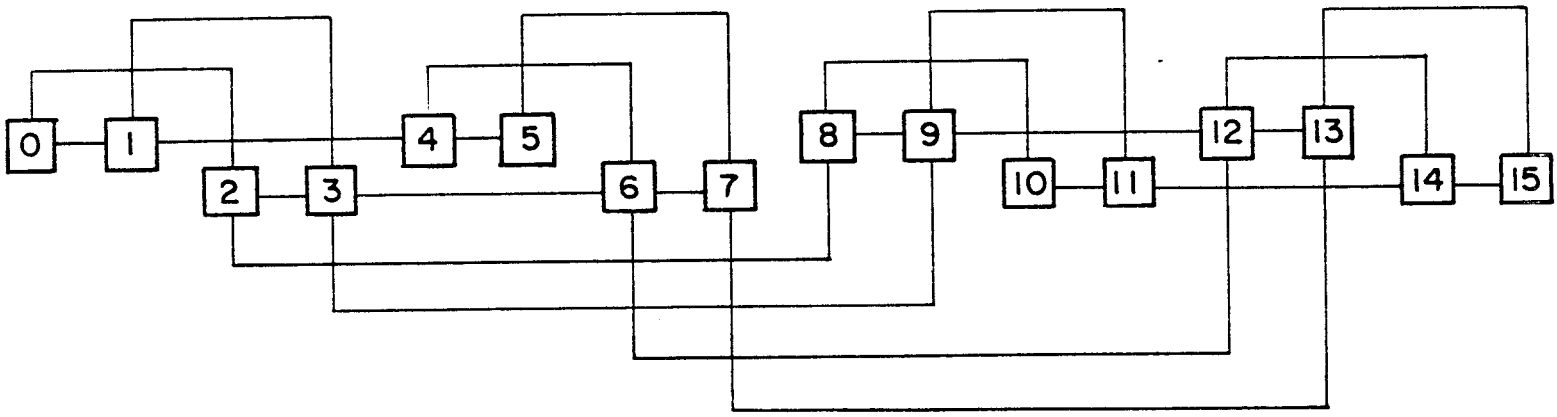

Figure 4-21: The linear form of the 16-cell mesh numbered by the shuffled row-major indexing scheme.

Lemma 20: An N-element sort can be performed on N cells interconnected with a mesh-based recirculation network in time $T = O(N^{1/2}t_R + (\log^2 N)t_C)$, if it takes t_R units of time to perform a unit-distance routing step and t_C units of time to perform a comparison-exchange operation.

Proof. (See [Thompson 77].) The time required to do a distance-2^k routing on the shuffled row-major indexing scheme for the N-cell mesh is just $2^{\lceil k/2 \rceil}t_R$. This observation may be verified easily for the case $N = 16$ of the right-hand mesh of

Figure 4-20. In the general case, it follows as a consequence of the way in which horizontal and vertical routings alternate in an ascending sequence or routing distances: distance-2^0 is one horizontal roure, distance-2^1 is one vertical route, distance-2^2 is two horizontal routes, distance-2^3 is two vertical routes, etc.

The time required for a 2^k-element merge (see Figure 4-5) is less than $8*2^{\lfloor (k-1)/2 \rfloor}t_R$. This bound is obtained from the fact that a 2^k-element merge requires two distance-2^{k-1} routings, two distance-2^{k-2} routings, . . . , and two distance-1 routings. The first four of these routings take at most $4*2^{\lfloor (k-1)/2 \rfloor}t_R$ time, and the times required by each of the succeeding groups of four routings form a geometric series with ratio one-half. The entire sum is thus less than twice that of the leading term, or $2*4*2^{\lfloor (k-1)/2 \rfloor}t_R$.

The time required for a 2^k-element sort (see Figure 4-6) on the shuffled row-major indexing scheme of the N-cell mesh is less than $32*2^{\lfloor (k-1)/2 \rfloor}t_R$. To verify this, note that a 2^k-element sort is formed of a 2^k-element merge, preceded by two 2^{k-1}-element merges running in parallel, preceded by four 2^{k-2}-element merges, . . . , preceded by $N/4$ four-element merges, preceded by $N/2$ two-element merges (these are the same as the two-element sorts that are actually specified in the construction). The first two of these together take less than $2*8*2^{\lfloor (k-1)/2 \rfloor}t_R$, by the bound of the preceding paragraph. The time taken by each succeeding pair of merges form a geometric series with ratio one-half. The entire 2^k-element sort thus takes less than $2*2*8*2^{\lfloor (k-1)/2 \rfloor}t_R$. (A more rigorous argument tightens this bound to $14*2^{k/2}t_R$ for even values of k [Thompson 77].)

The lemma is now almost proved, since an N-element sort ($N = 2^k$) takes routing time $O(2^{k/2}t_R) = O(N^{1/2}t_R)$. The comparison-exchange steps take time $O(\log^2 N t_C)$, for there are $O(\log^2 N)$ rows in the complete N-element bitonic sorting network. □

The performance of an N-element mesh-based sorting circuit can now be described.

Theorem 21: An N-element sort can be performed in $O(N \log^2 N)$ area and $O(N^{1/2} \log\log N)$ time on N comparison-exchange cells arranged in a square mesh.

Proof. The comparison-exchange cell of Lemma 16 and Figure 4-19 has a comparison-exchange time t_C of $O(\log N)$ and an area of $O(\log^2 N)$. If N of these are arranged in a mesh, they will occupy a total of $A = O(N \log^2 N)$ area. Attaching horizontal and vertical transceivers to each cell, as in Figure 4-11, will not affect the asymptotic area requirements, but will result in a data routing time t_R of $O(\log\log N)$. (As indicated in the discussion on page 92, t_R could be reduced to $O(1)$ by suitable changes in the upper bound model of computation.) Total time for an N-element sort is thus $T = O(N^{1/2} \log\log N)$, by Lemma 20. □

Note that this sorting circuit takes up as much area and is just as fast as the mesh-based FFT circuit of Subsection 4.3.2. As indicated by the following corollaries, it is very nearly optimal under the AT^2 metric of Chapter 3, for it is at most $O(\log\log^2 N)$ from the optimal $AT^2 = \Omega(N^2 \log^2 N)$. It is also nearly optimal under the AT^{2x} metric, since it is at most $O(\log\log^{2x} N)$ from the optimal value of $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$.

Corollary 22: $AT^2 = O(N^2 \log^2 N \log\log^2 N)$ for the N-element mesh-based sorting circuit.

Corollary 23: $AT^{2x} = O(N^{1+x} \log^2 N \log\log^{2x} N)$ for the N-element mesh-based sorting circuit.

4.5 Constant factors in the VLSI implementations

This thesis has demonstrated the existence of area-time tradeoffs in VLSI design, at least for chips that solve asymptotically large problems. It is now time to find out whether such tradeoffs will be available to designers in the foreseeable future. In other words, is a design with one hundred million transistors large enough for asymptotic analysis? The answer seems to be "almost": among designs that solve ten thousand element FFTs, the asymptotically "fast but large" shuffle-exchange-type chips are indeed faster but also slightly smaller than their "slow but small" mesh-type counterparts.

The arithmetic circuitry will be examined first. The multiply-add cell of Figure 4-13 or 4-17 can be built in about 10^4 units of area, if the word length is sixteen bits. This allows ample room for two 32-bit carry-save adders, a few hundred gates of control logic, and a few hundred bits of storage for constants. Accordingly, up to ten multiply-add cells would fit on a present-day chip with 10^5 units of area. By the year 1990, something like 10^4 cells could be formed on the 10^8 units of area available on a single silicon wafer. Such a circuit would perform a ten thousand element FFT.

The interconnections between the cells have yet to be considered. The mesh-type pattern poses no difficulties. Each of the multiply-add cells will be separated from its neighbor by a sixteen-wire bus (see Figure 4-11). Assuming that the 10^4-area cells are 10^2 units on a side, the width of the bus is almost negligible.

A ten thousand element FFT will also be feasible on a shuffle-exchange-type design. Here five thousand multiply-add cells should be laid out as tall, thin rectangles in a few rows at the bottom of a wafer (see Figure 4-17). The exchange connections are to nearest neighbors, and thus take up little area. The shuffle connections can be routed upwards to some unused horizontal "channel," across in this channel, then down to the appropriate cell. Since there are 10^4 horizontal channels on a wafer of 10^8 units of area, all shuffle connections can be made in this way.

The asymptotic size advantage of the mesh-based design is not apparent in chips of this size. In fact, the shuffle-exchange-type chip would actually be a little smaller, because it uses only half as many cells to perform a ten thousand element FFT. Furthermore, the routing control logic is much simpler for the shuffle-exchange circuit than for the mesh circuit, since the shuffle-exchange circuit uses the same recirculation pattern after each multiply-add step.

Mesh-type designs might become attractive from the standpoint of size if million-cell chips are ever feasible. One million multiply-add cells could be arranged in a

mesh-type pattern in 10^{10} units of area. However, shuffle connections among five hundred thousand cells would occupy 10^{11} square units of wiring in the best embedding known to the author.

As mentioned above, the shuffle-exchange design is a little faster than the mesh design for ten thousand element FFTs. In either case, there are $\log N = 13$ stages of computation in the FFT algorithm. Each multiply-add step takes about 10^2 clock periods, due primarily to the two *16*-bit by *32*-bit multiplications. Total multiply-add time for either design is thus 10^3 clock periods or a few microseconds, if a *5* nS clock is used [Mead 80].

The total routing time on a mesh-type design also turns out to be a few microseconds. By Equation (4.4), there are $4(10^2 - 1)$ unit distance routing steps in a ten thousand element FFT. Allowing two or three clock pulses per unit route for synchronization and buffering, routing takes 10^3 clock pulses or a few microseconds.

Routing on the shuffle-exchange design is somewhat faster than on the mesh-type design. In a ten thousand element FFT, there are *13* routing steps. Each step is a bit-serial transfer of *16* bits. Total routing time is thus several hundred clock periods or about one microsecond.

The speed advantage of the shuffle-exchange design will increase with the size of the FFT, as predicted by the asymptotic analysis. If a million-element FFT ever became feasible, the mesh-type design would perform $4(10^3 - 1)$ routing steps, while the shuffle-exchange design would do only several hundred steps. (The comparison is between $4N^{1/2}$ and $\log^2 N$.)

Therefore, ten thousand element FFT designs are just large enough for the asymptotic time analysis of this thesis, but not quite large enough for the area analysis.

Chapter 5
Conclusion

The main contributions of this thesis fall into four areas.

1. A new model of computation is developed, suited to the study of the area and time performance of VLSI chips.

2. A lower bound is obtained on the area A occupied by a graph when embedded in the plane, in terms of its minimum bisection width ω. For a k-level planar embedding, $A > \omega^2/4k^2$.

3. The informational complexity of a function is defined, determining the difficulty of computing that function on a VLSI chip.

4. Nearly tight upper and lower bounds are derived on achievable area-time performance for sorting and Fourier transformation, as summarized in the list and table below.

    - An N-element sorting or Fourier transformation problem can be solved on a chip of area $A = O(N \log^2 N)$ and time $T = O(N^{1/2} \log\log N)$, using a mesh-based interconnection scheme. This performance is nearly optimal for any AT^{2x} metric, $0 \leq x \leq 1$.

    - An N-element Fourier transformation problem can be solved on a chip of area $A = O(N^2/\log^{1/2} N)$ and time $T = O(\log^2 N)$, using an interconnection scheme based on the shuffle-exchange graph. This performance is optimal under the AT^2 metric.

    - An N-element sorting problem can be solved on a chip of area $A = O(N^2/\log^{1/2} N)$ and time $T = O(\log^3 N)$, using an interconnection scheme based on the shuffle-exchange graph. This performance is nearly optimal under the AT^2 metric.

	AT^2	AT^{2x}
Upper bounds:		
Mesh-based FFT	$O(N^2 \log^2 N \log\log^2 N)$	$O(N^{1+x} \log^2 N \log\log^{2x} N)$
Mesh-based sort	$O(N^2 \log^2 N \log\log^2 N)$	$O(N^{1+x} \log^2 N \log\log^{2x} N)$
Shuffle-exchange FFT	$O(N^2 \log^{7/2} N)$	$O(N^2 \log^{4x-1/2} N)$
Shuffle-exchange sort	$O(N^2 \log^{11/2} N)$	$O(N^2 \log^{6x-1/2} N)$
Lower bounds:		
(For any design)	$\Omega(N^2 \log^2 N)$	$\Omega(N^{1+x} \log^{2x} N)$

Table 5-1: Area-time complexity of sorting and Fourier transformation.

Several avenues of research have opened up as a result of this thesis.

The model of computation could be expanded to cover other possible VLSI technologies, such as Josephson junctions and magnetic bubble devices. This would permit formal development of upper and lower bound results for these technologies. Other modifications to the model would handle pipelined designs (page 84) and take full advantage of scaling (page 92).

Other problems should be studied besides sorting and Fourier transformation. It should be possible to derive fairly tight upper and lower bounds for matrix multiplication, integer multiplication, Gaussian elimination, and transitive closure.[5] The embedding of the shuffle-exchange graph described in Subsection 4.2.2

[5]Lower bounds were obtained for several of these problems as this thesis was being written. Savage [Savage 79b] proved lower bounds on boolean matrix multiplication and Gaussian elimination, using the VLSI model of computation. Brent and Kung [Brent 79] and Abelson [Abelson 80b] derived lower bounds for integer multiplication. Brent and Kung obtained a slightly more powerful result, for they were able to relax assumptions L6 and L7 of Chapter 2. In their model, inputs and outputs did not have to be contained on the chip; however, each input is available at its input port only once. They were able to prove that nearly all of the input bits had to be on the chip at the same time. The rest of their proof technique is similar to that used in this thesis.

deserves careful study. An improved result could tighten the gap between upper and lower bounds for designs using that interconnection pattern. (Dan Hoey and Charles Leiserson have obtained an $O(N^2/\log N)$ area embedding for the shuffle-exchange graph for the case that $N = 2^{2^n}$ [Hoey 80]. However, there is still an $O(\log N)$ gap between the upper and lower bound results.)

The definition of informational complexity could be improved. As it stands, it is demonstrably weak on such functions as "equality" and "comparison" (see page 66), despite its apparent strength on sorting and Fourier transformation. The problem seems to lie in its assumption of optimal coding (page 62).

Finally, the model of computation might be enhanced. It currently treats information as an imperishable item: in the view of the model, information cannot be destroyed, it merely flows from one side of the chip to another. This view is adequate to model the (nearly) one-to-one mappings of sorting and the DFT. However, it is sure to fail on exponentially hard functions. Somehow, the computation of a hard function must involve more than sending the inputs from one side of the chip to the other, yet this is enough to compute any function according to the present model.

Acknowledgements

To H. T. Kung, my thesis advisor, for his generous help and gentle guidance:

To Jon Bentley, for his patience in teaching a (sometimes ungracious) student to improve his technical writing, and for his suggestions on Theorems 3-1 and 3-2:

To Carver Mead, for his infectious vision of VLSI design:

To Michael Shamos, for his constructive feedback on the first versions of the VLSI model of computation:

To Bob Sproull, for trying to keep me honest in my descriptions of VLSI circuitry:

To all of the above, for serving on my thesis committee:

To Leo Guibas and the staff of Xerox PARC, for their support and advice during the earliest phases of this research:

To James B. Saxe and H. Wozniakowski, for their help with the original proofs of Lemmas 8 and 9:

--And last, but certainly not least--

To Charles Leiserson, for his ideas on VLSI modeling and his help on Theorem 2:

Thank you.

Appendix A
Control program for cell 0 of a mesh-based FFT Circuit

Timer	α_j	*Control*	*Comment*
			V registers contain initial data
$\log N$	--	$X_0 \leftarrow V$	Load X_0 register from V (This step omitted for cell i, $i < N/2$ i.e., omitted if $i_{(\log N)-1} = 1$)
			Begin stage 1
$(N^{1/2}/2)\ \text{loglog}\ N$	--	UP	Get data from cell $N/2$ (Cell i, $i \geq N/2$ will receive "garbage" during this routing, hence will compute garbage during this step)
$\log N$	--	$X_1 \leftarrow V$	Load register X_1
$\sim \log^2 N$	α^0	M-A	Multiply-add: $Y_0 \leftarrow X_0 + \alpha^0 X_1$, $Y_1 \leftarrow X_0 - \alpha^0 X_1$. (Other cells use different α_j values)
$\log N$	--	$V \leftarrow Y_1$	Send Y_1 down to cell $N/2$
$(N^{1/2}/2)\ \text{loglog}\ N$	--	DOWN	
$\log N$	--	$X_0 \leftarrow V$	Load X_0
			End stage 1

			Begin stage 2
$\log N$	--	$V \leftarrow Y_0$	
$(N^{1/2}/4) \log\log N$	--	UP	Receive data from cell $N/4$
$\log N$	--	$X_1 \leftarrow V$	Load X_1 (Cell i, $N/4 \leq i < N/2$ and $3N/4 \leq i < N$ will compute "garbage" during this stage)
$\sim\log^2 N$	α^0	M-A	
$\log N$	--	$V \leftarrow Y_1$	Send Y_1 down to cell $N/4$
$(N^{1/2}/4) \log\log N$	--	DOWN	
$\log N$	--	$X_0 \leftarrow V$	**End stage 2**
			Begin stage 3
$\log N$	--	$V \leftarrow Y_0$	
$(N^{1/2}/8) \log\log N$	--	UP	Receive data from cell $N/8$
$\log N$	--	$X_1 \leftarrow V$	
$\sim\log^2 N$	α^0	M-A	
$\log N$	--	$V \leftarrow Y_1$	Send Y_1 down to cell $N/8$
$(N^{1/2}/8) \log\log N$	--	DOWN	
$\log N$	--	$X_0 \leftarrow V$	**End stage 3**
			(Stages 4 through $(\log N)/2 - 1$ not shown)
			Begin stage $(\log N)/2$
$\log N$	--	$V \leftarrow Y_0$	
1	--	UP	Receive data from cell $N^{1/2} \log\log N$
$\log N$	--	$X_1 \leftarrow V$	
$\sim\log^2 N$	α^0	M-A	
$\log N$	--	$V \leftarrow Y_1$	Send Y_1 down to cell $N^{1/2} \log\log N$
1	--	DOWN	
$\log N$	--	$X_0 \leftarrow V$	**End stage** $(\log N)/2$
			Begin stage $(\log N)/2 + 1$
$\log N$	--	$H \leftarrow Y_0$	Note shift to horizontal routing
$(N^{1/2}/2) \log\log N$	--	LEFT	Receive data from cell $(N^{1/2}/2) \log\log N$
$\log N$	--	$X_1 \leftarrow H$	
$\sim\log^2 N$	α^0	M-A	
$\log N$	--	$H \leftarrow Y_1$	Send Y_1 over to cell $(N^{1/2}/2) \log\log N$
$(N^{1/2}/2) \log\log N$	--	RIGHT	
$\log N$	--	$X_0 \leftarrow H$	**end stage** $(\log N)/2 + 1$

			(Stages *(log N)/2 + 2* through *(log N) − 1* not shown)
			Begin stage *(log N)*
log N	--	$H \leftarrow Y_0$	
1	--	LEFT	Receive data from cell *1*
log N	--	$X_1 \leftarrow H$	
$\sim log^2 N$	α^0	M-A	
log N	--	$H \leftarrow Y_1$	Send Y_1 over to cell *1*
1	--	RIGHT	**End stage** *(log N)*
log N	--	$H \leftarrow Y_0$	This step omitted for cell *i*, *i* odd *H* registers contain transformed data

References

[Abelson 80a] Abelson, Harold.
Lower bounds on information transfer in distributed computations.
Journal of the ACM 27(2):384-392, April, 1980.

[Abelson 80b] Abelson, Harold and Andreae, Peter.
Information Transfer and Area-time Tradeoffs for VLSI Multiplication.
Communications of the ACM 23(1), January, 1980.

[Agarwal 75] Agarwal, Ramesh C. and Burrus, Sidney C.
Number theoretic transforms to implement fast digital convolutions.
Proc. IEEE 63(4):550-560, April, 1975.

[Aho 74] Aho, Alfred V. , Hopcroft, John E. and Ullman, Jeffrey D.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.

[Batcher 68] Batcher, K. E.
Sorting networks and their applications.
In *AFIPS Spring Joint Computer Conference*, pages 307-314. 1968.

[Bonneau 73] Bonneau, Richard J.
A Class of Finite Computation Structures Supporting the Fast Fourier Transform.
Technical Report 31, Massachusetts Institute of Technology Project MAC, March, 1973.

[Brent 79] Brent, R. P. and Kung, H. T.
The Area-Time Complexity of Binary Multiplication.
Technical Report CMU-CS-79-136, Carnegie-Mellon University, Computer Science Department, July, 1979.
(to appear in *JACM*).

[Cochran 67] Cochran, William T. , Cooley, James W. , *et al.*
What is the fast Fourier transform?.
IEEE Trans. on Audio and Electroacoustics AU-15(2):45-55, June, 1967.

[Cutler 78] Cutler, M. and Shiloach, Y.
Permutation layout.

Networks 8:253-278, 1978.

[Donath 79] Donath, W. E.
Placement and average interconnection lengths of computer logic.
IEEE Trans. Circuits and Systems CAS-26(4):272-277, April, 1979.

[Evans 79] Evans, Stephen A.
Scaling I^2L for VLSI.
IEEE Journal of Solid-State Circuits SC-14(2):318-326, April, 1979.

[Floyd 72] Floyd, Robert W.
Permuting information in idealized two-level storage.
In Raymond E. Miller and James W. Thatcher (editor), *Complexity of Computer Computations*, pages 105-109. Plenum Press, NY, 1972.

[Garey 74] Garey, M. R. , Johnson, D. S. and Stockmeyer, L.
Some simplified polynomial complete problems.
In *Proc. 6th Annual ACM Symposium on Theory of Computing*, pages 47-63. Assoc. Comp. Mach. , April, 1974.

[Grigoryev 76] Grigoryev, Y.
An application of separability and independence notions for proving lower bounds on circuit complexity.
Steklov Mathematical Institute, Leningrad Branch, 1976, pages 38-48.
(translation courtesy of J. E. Savage, Brown University).

[Hoey 80] Hoey, Dan and Leiserson, Charles E.
A layout for the shuffle-exchange network.
In *Proc. 1980 International Conf. on Parallel Processing.* IEEE Computer Society, August, 1980.

[Johannsen 78] Johannsen, Dave.
Hierarchical Power Routing.
Technical Report 2069, California Institute of Technology, October, 1978.

[Knuth 73] Knuth, Donald. E.
The Art of Computer Programming: Volume 3.
Addison-Wesley, 1973.

[Leiserson 80] Leiserson, Charles E.

Area-efficient graph layouts (for VLSI).
In *21st Annual Symp. on Foundations of Computer Science.* IEEE Computer Society, October, 1980.

[Linnik 44] Linnik, U. V.
On the least prime in an arithmetic progression. I. The basic theorem.
Mathemateceskii Sbornik (N. S.) 15(57):139-178, 1944.

[Lipton 77] Lipton, Richard J. and Tarjan, Robert E.
Applications of a planar separator theorem.
In *18th Annual Symp. on Foundations of Computer Science*, pages 162-170. IEEE Computer Society, October, 1977.

[Mead 79] Mead, Carver and Rem, Martin.
Cost and performance of VLSI computing structures.
IEEE Journal of Solid-State Circuits SC-14(2):455-462, April, 1979.

[Mead 80] Mead, Carver A. and Conway, Lynn A.
Introduction to VLSI Systems.
Addison-Wesley, 1980.

[Mohsen 79] Mohsen, Amr M. and Mead, Carver A.
Delay-time optimization for driving and sensing of signals on high-capacitance paths of VLSI systems.
IEEE Journal of Solid-State Circuits SC-14(2):462-470, April, 1979.

[Moravec 79] Moravec, Hans P.
Fully Interconnecting Multiple Computers with Pipelined Sorting Nets.
IEEE Trans. Comput. C-28(10):795-798, October, 1979.

[Moshell 76] Moshell, Michael and Rothstein, Jerome.
Parallel recognition of patterns: insights from formal language theory.
In *Proc. 1976 International Conf. on Parallel Processing*, pages 222-229. IEEE Computer Society, August, 1976.

[Nicholson 71] Nicholson, Peter J.
Algebraic theory of finite Fourier transforms.
Journal of Computer and System Sciences 5:524-547, 1971.

[Paterson 70] Paterson, M. S. and Hewitt, C. E.
Comparative schematology.
In *Project MAC Conf. on Concurrent Systems and Parallel Computation*, pages 119-127. Massachusetts Institute of Technology, June, 1970.

[Pease 68] Pease, Marshall C.
An adaptation of the fast fourier transform for parallel processing.
Journal of the ACM 15(2):252-264, April, 1968.

[Preparata 79] Preparata, Franco P. and Vuillemin, Jean.
The Cube-Connected Cycles: A Versatile Network for Parallel Computation.
In *20th Annual Symp. on Foundations of Computer Science*, pages 140-147. IEEE Computer Society, October, 1979.

[Rosenberg 79] Rosenberg, Arnold L.
On Embedding Graphs in Grids.
Technical Report RC 7559 (#2668), IBM T. J. Watson Research Center, February, 1979.

[Savage 77] Savage, John E. and Swami, Sowmitri.
Space-Time Tradeoffs in the FFT Algorithm.
Technical Report TRCS-31, Brown University, August, 1977.

[Savage 79a] Savage, John E. and Swamy, Sowmitri.
Space-time tradeoffs for oblivious integer multiplication.
In *6th Colloq. on Automata, Languages and Programming.* European Assoc. for Theoretical Computer Science, Springer-Verlag, July, 1979.

[Savage 79b] Savage, John E.
Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI Models.
Technical Report CS-50, Brown University Department of Computer Science, August, 1979.

[Seitz 79] Seitz, Charles L.
Self-timed VLSI systems.
In *Caltech Conference on VLSI*, pages 345-354. Caltech Computer Science Department, January, 1979.

[Shannon 49] Shannon, Claude E. and Weaver, Warren.
The Mathematical Theory of Communication.

Univ. of Illinois Press, Urbana, Ill. , 1949.

[Stevens 71] Stevens, J. E.
A Fast Fourier Transform Subroutine for Illiac IV.
Technical Report, Center for Advanced Computation, Illinois, 1971.

[Stone 71] Stone, Harold S.
Parallel processing with the perfect shuffle.
IEEE Trans. Comput. C-20(2):153-161, February, 1971.

[Sutherland 73] Sutherland, Ivan E. and Oestreicher, Donald.
How big should a printed circuit board be?.
IEEE Trans. Comput. C-22(5):537-542, May, 1973.

[Thompson 77] Thompson, C. D. and Kung, H. T.
Sorting on a mesh-connected parallel computer.
Communications of the ACM 20(4):263-271, April, 1977.

[Tompa 78] Tompa, Martin.
Time-space tradeoffs for computing functions, using connectivity properties of their circuits.
In *Proc. 10th Annual ACM Symp. on Theory of Computing*, pages 196-204. Assoc. Comp. Mach. , May, 1978.

[Valiant 76] Valiant, Leslie G.
Graph-theoretic properties in computational complexity.
Journal of Computation and Systems Sciences 13:278-285, 1976.

[von Neumann 66]
von Neumann, J.
The Theory of Self-Reproducing Automata.
Univ. of Illinois Press, Urbana, Ill. , 1966.

[Wagstaff 79] Wagstaff, Samuel S.
Greatest of the least primes in arithmetic progressions having a given modulus.
Mathematics of Computation 33(147):1073-1080, July, 1979.

[Winograd 76] Winograd, S.
On Computing the Discrete Fourier Transform.
Technical Report RC 6291 (#27013), IBM T. J. Watson Research Center, November, 1976.

[Yao 79] Yao, Andrew.

Some complexity questions related to distributive computing. In *Proc. 11th Annual ACM Symp. on Theory of Computing*, pages 209-213. Assoc. Comp. Mach. , April, 1979.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CMU-CS-80-140	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A COMPLEXITY THEORY FOR VLSI	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): C.D. THOMPSON	8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: AUGUST 1980
	13. NUMBER OF PAGES: 134
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)