Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing

CHARLES E. LEISERSON, MEMBER, IEEE

Abstract — **This paper presents a new class of universal routing networks called *fat-trees,* which might be used to interconnect the processors of a general-purpose parallel supercomputer. A fat-tree routing network is parameterized not only in the number of processors, but also in the amount of simultaneous communication it can support. Since communication can be scaled independently from number of processors, substantial hardware can be saved over, for example, hypercube-based networks, for such parallel processing applications as finite-element analysis, but without resorting to a special-purpose architecture.**

Of greater interest from a theoretical standpoint, however, is a proof that a fat-tree of a given size is nearly the best routing network of that size. This *universality theorem* is proved using a three-dimensional VLSI model that incorporates wiring as a direct cost. In this model, hardware size is measured as physical volume. We prove that for any given amount of communications hardware, a fat-tree built from that amount of hardware can simulate every other network built from the same amount of hardware, using only slightly more time (a polylogarithmic factor greater). The basic assumption we make of competing networks is the following. In unit time, at most $O(a)$ bits can enter or leave a closed three-dimensional region with surface area a. (This paper proves the universality result for *off-line* simulations only.)

Index Terms — **Fat-trees, interconnection networks, parallel supercomputing, routing networks, universality, VLSI theory.**

I. Introduction

MOST routing networks for parallel processing supercomputers have been analyzed in terms of performance and cost. Performance is typically measured by how long it takes to route permutations, and cost is measured by the number of switching components and wires. This paper presents a new routing network called fat-trees, but analyzes it in a somewhat different model. Specifically, we use a three-dimensional VLSI model in which *pin boundedness* has a direct analog as the bandwidth limitation imposed by the surface of a closed three-dimensional region. Performance is measured by how long it takes to route an arbitrary set of messages, and cost is measured as the volume of a physical implementation of the network. We prove a *universality* theorem which shows that for a given volume of hardware, no network is much better.

Unlike a computer scientist's traditional notion of a tree, fat-trees are more like real trees in that they get thicker further from the leaves. In physical structure, a fat-tree resembles, and is based on, the *tree of meshes* graph due to Leighton [12], [14]. The processors of a fat-tree are located at the leaves of a complete binary tree, and the internal nodes are switches. Going up the fat-tree, the number of wires connecting a node with its father increases, and hence the communication bandwidth increases. The rate of growth influences the size and cost of the hardware as well.

Most networks that have been proposed for parallel processing are based on the Boolean hypercube, but these networks suffer from wirability and packaging problems and require nearly order $n^{3/2}$ physical volume to interconnect n processors. In his influential paper on "ultracomputers" [27], Schwartz demonstrates that many problems can be solved efficiently on a supercomputer-based on a shuffle network [28]. But afterwards, Schwartz comments, "The most problematic aspect of the ultracomputer architecture suggested in the preceding section would appear to be the very large number of intercabinet wires which it implies." Schwartz then goes on to consider a "layered" architecture, which seems easier to build, but which may not have all the nice properties of the original architecture.

On the other hand, there are many applications that do not require the full communication potential of a hypercube-based network. For example, many finite-element problems are planar, and planar graphs have a bisection width of size $O(\sqrt{n})$, as was shown by Lipton and Tarjan [19]. Moreover, any planar interconnection strategy requires only $O(n)$ volume. Thus, a natural implementation of a parallel finite-element algorithm would waste much of the communication bandwidth provided by a hypercube-based routing network.

Fat-trees are a family of general-purpose interconnection strategies which effectively utilize any given amount of hardware resource devoted to communication. This paper proves that for a given physical volume of hardware, no network is much better than a fat-tree. Section II introduces fat-tree architectures and gives the logical structure of one feasible implementation. Section III shows how communication on a fat-tree can be scheduled off-line in a near-optimal fashion. Section IV defines the class of *universal* fat-trees and investigates their hardware cost in a three-dimensional VLSI model. Section V contains several combinatorial theorems concerning the recursive decomposition of an arbitrary routing network, and Section VI uses these results to demonstrate that fat-trees are indeed a class of hardware-efficient universal routing networks. Finally, Section VII offers some remarks about the practicality of fat-trees.

Manuscript received February 1, 1985; revised May 30, 1985. This work was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622. A preliminary version of this paper was presented at the IEEE 1985 International Conference on Parallel Processing, St. Charles, IL, Aug. 1985.

The author is with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

II. Fat-Trees

This section introduces fat-trees as a routing network for parallel computation. The parallel computer based on fat-trees that we present is somewhat arbitrary and is influenced by the various connection machine projects [5], [9], [11] conceived at M.I.T. The computational model is not meant to be exclusive — the results in this paper undoubtedly apply to more general models. Moreover, arbitrary "engineering design decisions," which may not be the best choices from either a practical or a theoretical perspective, have been made in this description of fat-trees. Most of the choices influence the results by only a logarithmic factor, however, and do not affect the overall thrust of the paper — the universality theorem in Section VI.

The intuitive model for parallel computation that we use is a parallel computation engine composed of a set of processors interconnected by a *routing network*. The processors share no common memory, and thus they must communicate through the routing network, using *messages*. The job of the routing network is to see that all messages eventually reach their destinations as quickly as possible.

A fat-tree FT is a routing network based on a complete binary tree. (See Fig. 1.) A set P of n processors is located at the leaves of the fat-tree. Each edge of the underlying tree corresponds to two *channels* of the fat-tree: one from parent to child, the other from child to parent. Each channel consists of a bundle of wires, and the number of wires in a channel c is called its *capacity*, denoted by cap(c). The capacities of channels in the routing network are determined by how much hardware we can afford, a topic to be discussed in Section IV. The channel leaving the root of the tree corresponds to an interface with the external world. Each (internal) node of the fat-tree contains circuitry that switches messages between incoming channels and outgoing channels.

Messages produced by processors are *batched* into *delivery cycles*. During a delivery cycle, a processor may send messages through the network to other processors. Some messages may be lost in the routing network during a delivery cycle. Thus, in general, at the end of the delivery cycle, acknowledgments are sent from the destination processor back to the source processor. Messages that are not delivered must be sent again in subsequent delivery cycles.

The nodes of the fat-tree accomplish most of the switching. In order to understand their function, one must first understand how the routing of messages is accomplished. A *message set* $M \subseteq P \times P$ is a set of *messages* (i, j). If $(i, j) \in M$, then processor i has a message to be sent to processor j. (We omit details concerning the contents of messages and the handling of messages routed to and from the external interface.) Routing in the fat-tree is basically easy since every message has a unique path in the underlying complete binary tree. A message going from processor i to processor j goes up the tree to their least common ancestor and then back down according to the least significant bits of j. Notice that at any node of the fat-tree, there are only two choices for the routing of a message. If it comes into a node from a left subtree, for example, it can only go up or down to the right. Thus, a bit string of length at most 2 lg n is sufficient to represent the destination of any message.[1]

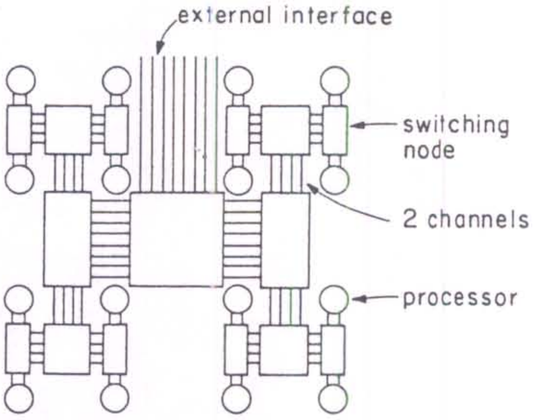

Fig. 1. The organization of a fat-tree. Processors are located at the leaves, and the internal nodes contain concentrator switches. The capacities of channels increase as we go up the tree.

We shall consider communication through the fat-tree network to be synchronous and bit serial. Messages snake through the tree with leading bits of a message establishing a path for the remainder to follow. Since some of the paths through the tree are longer than others, synchronization of the departures and arrivals of messages can be a bit tricky. Buffering of messages by the sending processors is one solution to this problem. (As was mentioned before, there are many other engineering alternatives that lead to the same kinds of theoretical results reported here.) The differing lengths of paths in the fat-tree are actually a major advantage of the network because messages can be routed locally without soaking up the precious bandwidth higher up in the tree, much as telephone communications are routed within an exchange without using more expensive trunk lines.

The messages in the network obey the bit-serial protocol shown in Fig. 2. The first bit is the *M bit*, which tells whether the remaining bits actually contain a message. Next come the *address* bits, which name the destination processor. The final field in the message format is the data themselves. As messages are routed through the network, each node uses the M bit to identify whether a wire carries a message, and it uses the first address bit to make a routing decision. A path is established through the node for a new M bit and the remaining message bits to follow. The address bits are stripped off one by one as the message establishes a path through the network.

A fat-tree node has three input ports, U_I, L_I, and R_I, and three output ports, U_O, L_O, and R_O, connected in the natural way to the wires in the channels. Messages entering input port L_I will go either to output port U_O or to output port R_O. The logic of the switching circuitry in a node consists of three similar portions, shown in Fig. 3. A wire from an input port is fanned out towards the two opposite output ports. The M bit of each wire is then examined to determine whether the wire has a message. On the next clock tick, the first address bit is examined on both branches of the input wire. By ANDing

[1] We use the notation lg n to mean $\max\{1, \log_2 n\}$.

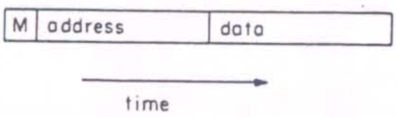

Fig. 2. The format of bit-serial messages. The first bit that a switch sees is the M bit, which indicates whether an input wire actually contains a message. The address bits arrive bit-serially in subsequent time steps, and the message contents are last.

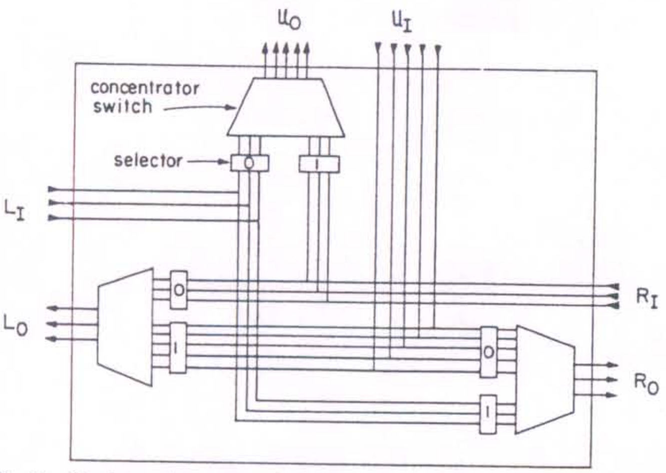

Fig. 3. The internal structure of a fat-tree node. A selector determines which messages are destined for an output port, and then a concentrator switch establishes disjoint electrical paths for as many of those messages as possible.

the M bit with either the address bit or its complement, an M bit is determined for each branch by a *selector*. Next, the messages destined for an output port, which currently occupy many wires, are switched onto fewer wires by a *concentrator switch*.

The job of the concentrator switch is to create electrical paths from those input wires that carry messages to fewer output wires. Obviously, if there are more input messages than output wires, some messages will be lost. In this case we shall say that the output channel is *congested*. We have already mentioned an acknowledgment mechanism that detects when messages are lost due to congestion.

For the time being, we shall assume that the concentrator switch has the following property. If there is no congestion—that is, the number of input messages does not exceed the number of output wires—then no messages are lost. The concentrator switch that we shall present in Section IV is a *partial concentrator* and does not have exactly this property, but it makes little difference to the theoretical results. This circuit has $O(m)$ components if there are a total of m incident wires, and it switches in constant time. Thus, the time required for an entire delivery cycle in a fat-tree of n processors is $O(\lg n)$.

Although we have described the general setting as an *on-line* switching environment, this paper makes the simplifying assumption that the fat-tree nodes contain *off-line* circuitry, in that the switches, although dynamically set, have their settings predetermined by an off-line scheduling algorithm. Naturally, it would be better to dynamically determine the settings themselves in real time, and indeed, it is possible to build such on-line switches, but these results will be reported elsewhere [8]. We have chosen here to prove the weaker off-line results so as to simplify the presentation of the universality theorem in Section VI.

There are several consequences of the off-line assumption that bear mention, however. For example, the results apply to practical situations when the settings of switches can be "compiled," as when simulating a large VLSI design or emulating a fixed-connection network. Also, some of the mechanisms—such as acknowledging the receipt of messages, which is necessary in an on-line environment—can be omitted from the off-line hardware structure, thereby reducing the complexity of the design.

III. Off-Line Scheduling on Fat-Trees

The concentrator switches in the nodes of a fat-tree routing network guarantee that no messages are lost unless there is congestion. This section gives an algorithm for scheduling the delivery of an arbitrary set of messages so that all messages will be delivered. We give a simple value, called the *load factor* of a set of messages, which provides a lower bound on how quickly the messages can be delivered. We show that for an arbitrary message set, off-line scheduling can be done optimally to within a logarithmic factor of the number of processors.

Let us be more precise about the off-line scheduling problem. Let FT be a fat-tree on n processors, and let C be the set of channels in FT. For any channel $c \in C$, the capacity $\text{cap}(c)$, which is the number of wires in the channel, is also the maximum number of simultaneous messages the channel can support because we are assuming bit-serial communication. Since each message between two processors determines a unique path in the underlying complete binary tree, we can define $\text{load}(M, c)$ to be the total number of messages in a message set M that must go through channel c. We call M a *one-cycle* message set if $\text{load}(M, c) \le \text{cap}(c)$ for all channels $c \in C$. If all capacity constraints are met, a fat-tree with ideal concentrator switches can route every message in one delivery cycle.

A *schedule* of a message set M is a partition of M into one-cycle message sets $M_1, M_2, \cdots, M_d$ where d is the total number of delivery cycles. A simple lower bound on d for an arbitrary message set M is $d \ge \max_c(\text{load}(M, c)/\text{cap}(c))$, which leads to the following definition.

Definition: Let M be a message set, and let $c \in C$ be a channel in a fat-tree. The *load factor* $\lambda(M, c)$ of a channel c due to M is

$$\lambda(M, c) = \frac{\text{load}(M, c)}{\text{cap}(c)},$$

and the *load factor* of the entire fat-tree due to M is

$$\lambda(M) = \max_{c \in C} \lambda(M, c).$$

A message set M is a *one-cycle* message set if $\lambda(M) \le 1$.

The simple lower bound on the number d of delivery cycles required for any schedule of M can now be reexpressed as $d \ge \lambda(M)$. The next theorem shows that this lower bound can be achieved to within a logarithmic factor of n.

Theorem 1: Let FT be a fat-tree on n processors, and let C be the set of channels in FT. Then for any message set M *with $\lambda(M) > 1$, there is an off-line schedule $M_1, M_2, \cdots, M_d$ such that $d = O(\lambda(M) \lg n)$.*

Proof: The idea is to partition the messages going from left to right through the root of the fat-tree into at most $2\lambda(M)$ one-cycle message sets, to do the same for the messages going from right to left, and then to recursively partition the messages in the two subtrees of the root. Let Q_1 be the subset of M consisting of those messages that must go through the root from left to right. The scheduling algorithm will begin by partitioning Q_1 into two message sets Q_2 and Q_3. It then iteratively refines each Q_k into Q_{2k} and Q_{2k+1}, until each Q_k, $k = r, \cdots, 2r-1$ is a one-cycle message set for some $r \le 2\lambda(M)$. The r message sets $Q_r, \cdots, Q_{2r-1}$ form the initial sequence of the schedule.

The algorithm similarly partitions the message set consisting of messages going from right to left in the fat-tree and adds them to the schedule. (Each of these message sets can, in fact, be routed at the same time as one of the Q_k.) Finally, the algorithm recursively partitions the messages remaining within the two subtrees of the root. The upper bound of $2\lambda(M)$ one-cycle message sets holds for all messages routed through the root of a subtree. But since all subtrees with roots at the same level can be routed at the same time, the total number of delivery cycles required is at most the height of the fat-tree times the time for one level, which yields $d = O(\lambda(M) \lg n)$.

It remains to show that the message sets can be partitioned effectively. Consider once again the message set Q_1 of messages going left to right through the root of the fat-tree. We now show that each message set Q_k, $k = 1, 2, \cdots, r-1$, can be partitioned into Q_{2k} and Q_{2k+1} so that for every channel $c \in C$, the messages of Q_k that go through c are split exactly evenly, that is, so that $\text{load}(Q_{2k}, c) \le \lceil (1/2)\, \text{load}(Q_k, c) \rceil$ and $\text{load}(Q_{2k+1}, c) \le \lceil (1/2)\, \text{load}(Q_k, c) \rceil$. The partitioning consists of two parts, matching and tracing, and is reminiscent of switch setting in a Benes network [34] and the Eulerian tour routing algorithm from [10].

First, do the matching. Consider each message in Q_k as being a string with two ends: a source end and a destination end. Within each processor, match as many pairs of string ends as possible until at most one message of Q_k is unmatched within each processor. Notice that source ends are matched only with source ends and destination ends only with destination ends because all messages in Q_k go left to right through the root. Then consider two-leaf subtrees. If each of the two leaves has one unmatched string end, match the ends. Continue matching the unmatched string ends in four-leaf subtrees, and so on up the fat-tree. At every level of the fat-tree, at most one string end is unmatched in each of the two subtrees of a node. At the root, at most one string end from each side will be unmatched (when there is an odd number of messages going from left to right through the root).

Now the tracing phase begins. If there is an unmatched string end in the left subtree, start with it. Otherwise, pick a string end arbitrarily from the left subtree. Put the corresponding message into Q_{2k}, and follow the string to the right subtree. Find the mate of the string end on the right side, and put the corresponding message into Q_{2k+1}. Follow this new string back to the left side, find its mate, and put the corresponding message into Q_{2k}. In general, when traversing a string left to right, put the corresponding message into Q_{2k}. When traversing right to left, put the message into Q_{2k+1}. If we discover that a string end has no mate, or that the message corresponding to the mate has already been assigned, we have either found the (one) unmatched string end on the right or completed a cycle. In either event, pick another string end arbitrarily and continue until all messages in Q_k have been assigned either to Q_{2k} or to Q_{2k+1}.

To see that this algorithm evenly splits the messages of Q_k in every channel c, observe that the number of times we enter a subtree of the fat-tree is equal to the number of times we leave, unless we are tracing the one possible string end matched outside the subtree. Since the split is even in every channel, the partitioning of Q_1 into one-cycle message sets $Q_r, \cdots, Q_{2r-1}$ will be achieved when

$$\begin{aligned} r &\le 2 \max_c \frac{\text{load}(Q_1, c)}{\text{cap}(c)} \\ &\le 2 \max_c \frac{\text{load}(M, c)}{\text{cap}(c)} \\ &\le 2\lambda(M), \end{aligned}$$

which completes the proof. ■

For the special case when $\text{cap}(c) > a \lg n$, for some $a > 1$, the logarithmic factor in the upper bound of Theorem 1 can be removed. Thus, under these conditions, the lower bound of the load factor can be met almost exactly.

Corollary 2: Let FT be a fat-tree on n processors, let C be the set of channels in FT, and suppose that there is a constant $a > 1$ such that $\text{cap}(c) \ge a \lg n$ for all $c \in C$. Then for any message set M, there is an off-line schedule $M_1, M_2, \cdots, M_d$ such that $d = O((a/a - 1)\lambda(M))$.

Proof: For each channel $c \in C$, define a set of *fictitious capacities* $\text{cap}'(c) = \text{cap}(c) - \lg n$. The fat-tree with the fictitious capacities has a load factor $\lambda'(M) \le (a/a - 1)\lambda(M)$. Now use the scheduling algorithm of Theorem 1, but during the recursion on lower levels of the tree, rather than using new message sets, simply reuse the $2\lambda'(M)$ message sets produced by partitioning the messages through the root.

The bisections at a given level produce partitions of the set of messages that are equal to within one, and this error can accumulate in a single channel as we go down the tree. The largest value of the error can be as much as $\lg n$, but the actual capacities are never exceeded, and so each of the $2\lambda'(M)$ message sets will be routable in one delivery cycle. ■

Thus, for example, if the capacities are each at least $2 \lg n$, the number of delivery cycles is not worse than $4\lambda(M)$. (In fact, the divide-and-conquer partitioning of messages can be improved to $2\lambda(M) + o(\lambda(M))$.)

IV. The Hardware Requirements of Fat-Trees

This section investigates the amount of hardware required by a fat-tree. We give a precise description of how the switches in the nodes of a fat-tree might be implemented and determine how much hardware a node requires. We then define the channel capacities of *universal* fat-trees. Finally, we determine the amount of hardware required to build universal fat-trees.

The model for hardware that we use is an extension of Thompson's two-dimensional VLSI model [29] by making the natural extension to three dimensions. In this model, wires occupy volume and have a minimum cross-sectional area. Similar three-dimensional models have been studied by Rosenberg [26] and Leighton and Rosenberg [16].

We first present an implementation of a fat-tree node. As was shown in Fig. 3, most of the switching components are contained in the three concentrator switches. According to the three-dimensional VLSI model, however, we must also be concerned with the amount of wire consumed by the interconnection of the components. We shall show that a fat-tree node with m incident wires can be built with $O(m)$ components in a box whose side lengths are $O(h\sqrt{m})$, $O(h\sqrt{m})$, and $O(\sqrt{m}/h)$, for any $1 \le h \le \sqrt{m}$. The node requires constant time to route its inputs.

We shall need some definitions. An (r, s) *concentrator graph* [21] is a directed acyclic graph with r inputs and $s \le r$ outputs such that any $k \le s$ inputs can be simultaneously connected to some k outputs by vertex-disjoint paths. An (r, s, α) *partial concentrator graph* is a directed acyclic graph with r inputs and $s \le r$ outputs and a constant $0 < \alpha < 1$ such that any $k \le \alpha s$ inputs can be simultaneously connected to some k outputs by vertex-disjoint paths.

Pinsker [21] and Pippenger [22] showed that (r, s) concentrator networks can be built with $O(r)$ components using probabilistic constructions, but they do not bound the depth of the graph, which we wish to be constant. Pippenger [23], however, uses another probabilistic argument to construct (r, s, α) partial concentrator graphs for sufficiently large r where $s = 2r/3$ and $\alpha = 3/4$. The partial concentrator graphs are bipartite (no intermediate vertices between inputs and outputs), every input has degree at most 6, and every output has degree at most 9. By pasting several of these graphs together, outputs to inputs, any constant ratio of concentration can be obtained in constant depth. For a given set of inputs, the paths through the graph can be set up in polynomial time using network flow techniques or by performing a sequence of matchings on each level of the graph.

We use a partial concentrator graph to construct a good concentrator switch. We simply make switching decisions at the inputs to each level. These decision bits can be interleaved with the address bits that specify the path of a message through the fat-tree. In order to use the off-line routing results from Section III, we treat the actual capacity of a channel as α times the number of wires, which changes the results by only a constant factor.

We now turn our attention to the physical structure of a fat-tree node. A node with m incident wires contains $O(m)$ components. The next theorem gives the physical volume necessary to wire the components.

Lemma 3: A set of m components and external wires can be wired together according to an arbitrary interconnection pattern to fit in a box whose side lengths are $O(h\sqrt{m})$, $O(h\sqrt{m})$, and $O(\sqrt{m}/h)$, for any $1 \le h \le \sqrt{m}$.

Proof: We need to use the fact that in two dimensions, any permutation of m inputs and m outputs can be routed in $O(m^2)$ area, which can be seen by considering a "crossbar" layout. Thus, in two dimensions, the wiring of the components and external wires can be accomplished by laying all components and external wires along a line and routing the permutation dictated by the interconnection.

The construction in three dimensions is essentially that of Leighton and Rosenberg [16]. In three dimensions, the external wires and components lie on a face of a box. Any permutation of m inputs and m outputs can be routed in a box of $O(m^{3/2})$ volume where each side has length $O(\sqrt{m})$. This proves the result of the theorem for constant h.

To extend the result, we use a result of Thompson [29] on converting a layout of height h into a layout of height 2. Consider slicing the box into slices of height h, and consider one such slice. If we expand each of the other two dimensions by a factor of h, the h layers can be superimposed, slightly offset from one another. Since this can be done with each of the slices simultaneously, the theorem follows. ■

We are now in a position to ascertain the cost of a fat-tree implementation based on the capacities of its channels. If the capacities of the fat-tree channels are determined arbitrarily, the analysis could be messy. For the fat-trees that will be used in universality results of Section VI, however, the channel capacities can be characterized by the capacity at the root. This section defines the channel capacities of a *universal* fat-tree and evaluates the hardware costs of an implementation. Without loss of generality, and for simplicity, we assume in this section that the number of connections to each processor in the fat-tree is 1.

Let FT be a fat-tree on n processors, and let C be the set of channels in FT. Consider each node to have a level number that is its distance to the root, and give each channel $c \in C$ the same level number as the node beneath it. Thus, for example, the root and the channel between the root and the external interface are both at level 0. The processors and the channels leaving them are at level lg n. If the channel at level 0 has capacity w, then we say that FT has *root capacity* w.

Definition: Let FT be a fat-tree on n processors with root capacity w where $n^{2/3} \le w \le n$. Then if each channel $c \in C$ at level k satisfies

$$\operatorname{cap}(c) = \min\left\{\left\lceil \frac{n}{2^k} \right\rceil, \left\lceil \frac{w}{2^{2k/3}} \right\rceil\right\},$$

we call FT a *universal fat-tree*.

The capacities of the channels of a universal fat-tree grow exponentially as we go up the tree from the leaves. Initially, the capacities double from one level to the next, but at levels closer than $3\,\lg(n/w)$ to the root, the channel capacities grow at the rate of $\sqrt[3]{4}$.

We can now determine the hardware required by a universal fat-tree.

Theorem 4: Let FT be a universal fat-tree on n processors with root capacity w where $n^{2/3} \le w \le n$. Then there is an implementation of FT in a cube of volume $v = O((w\,\lg(n/w))^{3/2})$ with $O(n\,\lg(w^3/n^2))$ components.

Proof: We first establish the component count. For a node at level $k \le 3\,\lg(n/w)$, the number of components in the node is $O(w/2^{2k/3})$, and the number of nodes at level k is 2^k. Thus, the number of components in all levels between 0 and $3 \lg(n/w)$ is

$$\sum_{k=0}^{3\lg(n/w)} 2^k O(w/2^{2k/3}) = w \sum_{k=0}^{3\lg(n/w)} O(2^{k/3}) = O(n)$$

since the largest term of the geometric series occurs when $k = 3 \lg(n/w)$. Nearer the leaves, each level has about the same number of components. The total number in the levels between $3 \lg(n/w)$ and $\lg n$ is

$$\sum_{k=3\lg(n/w)}^{\lg n} 2^k O(n/2^k) = O(n \lg(w^3/n^2)) .$$

Thus, the number of components nearer the leaves of the fat-tree dominates.

The volume bound is somewhat more intricate to establish, but is essentially the unrestricted three-dimensional layout construction given by Leighton and Rosenberg [16]. The interested reader is referred to their paper. Similar divide-and-conquer layout strategies for two dimensions can be found in [3], [12], [14], [17], [18], [32]. ■

Theorem 4 gives the volume of a fat-tree in terms of its root capacity. For the universality results of Section VI, we shall be interested in the reverse.

Definition: Let FT be a universal fat-tree that occupies volume v and has root capacity $\Theta(v^{2/3}/\lg(n/v^{2/3}))$. Then FT is a *universal fat-tree of volume* v.

Remark: A universal fat-tree on n processors of volume v must satisfy $v = \Omega(n \lg n)$ and $v = O(n^{3/2})$ to be well defined. By modifying the definition of a universal fat-tree, the lower bound can be relaxed to $\Omega(n)$, which results in minor changes to the bounds quoted in the universality theorem of Section VI.

V. Decomposition Trees

The physical implementation of a routing network constrains the ability of processors in a parallel supercomputer to communicate with one another. The universality theorem from Section VI makes essentially one assumption about competing networks: at most $O(a)$ bits can pass through a surface of area a in unit time. This assumption can be brought to bear on an arbitrary portion of a routing network implementation through the use of *decomposition trees,* a refinement of the graph-theoretic notion of *separators* [19]. Similar results can be found in the VLSI theory literature. The results presented here generalize and greatly simplify some of the constructions in the literature, notably those in [3], [4], and [13]. The generalizations are necessary for the proof of the universality theorem.

A routing network R interconnecting a set P of processors has a $[w_0, w_1, \cdots, w_r]$ *decomposition tree* if the amount of information that can enter or leave the set P of processors from the outside world is at most w_0 bits per unit time; P can be partitioned into two sets P_0 and P_1 such that the amount of information that can enter or leave each set is at most w_1 bits per unit time; each of P_0 and P_1 can be partitioned into two sets such that the bandwidth to and from each of the four sets is at most w_2; and so on, until every set at the rth level has either zero or one processors in it. When the bandwidth decreases by a constant amount from one level to the next, we shall adopt a shorthand notation. We shall say that R has a (w, α) decomposition tree for $1 < \alpha \leq 2$ if it has a $[w, w/\alpha, w/\alpha^2, \cdots, O(1)]$ decomposition tree. (For VLSI graph layouts, there is a similar notion called *bifurcators* [3], [13].)

Theorem 5: Let R be a routing network that occupies a cube of volume v. Then R has an $(O(v^{2/3}), \sqrt[3]{4})$ decomposition tree.

Proof: The cube has side length $\sqrt[3]{v}$ and surface area $6v^{2/3}$. Imagine a rectilinearly oriented plane that splits the cube into two equal boxes, each occupying volume $v/2$. This cutting plane naturally partitions the processors into two sets. Partition each of the two boxes by repeating this procedure with a plane perpendicular to the first. Continuing now in the third dimension yields eight cubes. Repeat this procedure until each box contains either zero or one processors.

The volume of each of the 2^i boxes generated by the ith cut is $v/2^i$, and the surface area is at most $4\sqrt[3]{4}\,(v/2^i)^{2/3}$. Let γ be the constant factor by which the bandwidth of information transfer differs from the surface area. Then the routing network R has a $(4\sqrt[3]{4}\,\gamma v^{2/3}, \sqrt[3]{4})$ decomposition tree. ■

A decomposition tree generated by the cutting plane method can be unbalanced in the sense that the number of processors lying on either side of a given cut may be unequal. Following the approach of Bhatt and Leighton [3], we define a *balanced decomposition tree* to be a decomposition tree in which the number of processors on either side of a given partition is equal, to within one. We shall show that a balanced decomposition tree can be produced from an unbalanced one.

First, however, we shall need two combinatorial lemmas. The first, which deals with the partitioning of strings of pearls, is typical of lemmas proved in the VLSI theory literature.

Lemma 6: Consider any two strings composed of even numbers of black and white pearls. By making at most two cuts, the pearls can be divided into two sets, each containing at most two strings, such that each set has exactly half the pearls of each color.

Proof:[2] Call the strings L and S for "long" and "short." We use a continuity argument to show that two sets A and $\overline{A}$ satisfying the conditions of the lemma can always be produced. Place the strings L and S end-to-end in a circle, as is illustrated in Fig. 4(a). Let A be the set of pearls comprising the shaded half of the circle in Fig. 4(b), and let $\overline{A}$ be the set of pearls in the other half. Suppose without loss of generality that the set A contains too many black pearls and set $\overline{A}$ contains too few. We shall show how to transform set A so that it occupies the initial position of set $\overline{A}$. The transformation consists of a sequence of moves such that for each

[2] Thanks to G. Miller of USC, who provided this argument, which is simpler than our original algebraic proof.

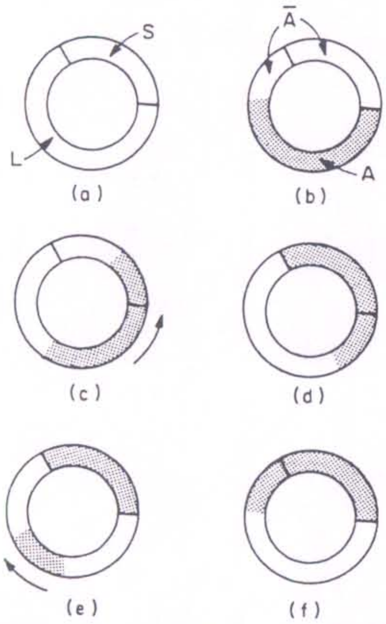

Fig. 4. The partitioning argument. (a) The two strings L and S laid end-to-end in a circle. (b) The initial position of set A. (c)–(f) The transformation of A into $\overline{A}$.

move, the number of blacks within set A changes by at most one. Since set A starts out with too many black pearls and ends with too few, by continuity there will be a position in the middle where A has exactly half the black pearls. Furthermore, because A has half the total number of pearls, it will also have half the white pearls.

The transformation begins by rotating set A counterclockwise, as shown in Fig. 4(c), until it reaches the position shown in Fig. 4(d). Then, set A is broken into two pieces and the tailing piece is rotated clockwise until it meets up with the leading piece on the other side, shown in Fig. 4(e) and (f). The position of set A is now the initial position of set $\overline{A}$. As can be verified, at all times during the transformation, sets A and $\overline{A}$ each contain at most two strings. ∎

Lemma 7: Let T be a complete binary tree drawn in the natural way with leaves on a straight line, and consider any string s of k consecutive leaves. Then there exists a forest F of complete binary subtrees of T such that 1) the leaves of F are precisely the leaves in s, 2) there are at most two trees of any given height, and 3) the height of the largest tree is at most $\lg k$.

Proof: The forest is constructed from the maximal complete subtrees of T whose leaves lie only in s. ∎

Theorem 8: Let R be a routing network on n processors that has a $[w_0, w_1, \cdots, w_r]$ decomposition tree T. Then R has a $[w'_0, w'_1, \cdots, w'_{\lceil \lg n \rceil}]$ balanced decomposition tree T' where

$$w'_i = 4 \sum_{k=i}^{r} w_k .$$

Proof: Draw the decomposition tree T in the natural way with the 2^r leaves on a line. Each leaf either contains a processor or else it is empty. If the leaf contains a processor, color it black; otherwise, color it white. Considering the line of processors as a string of black and white pearls, as in Lemma 6, we can cut the string in at most two places such that the pearls are divided into two sets, each containing at most two strings, such that each set has exactly half the pearls of each color. This partition represents the first level in the balanced decomposition tree T'.

Recursively partition each of the two sets using Lemma 6. At each step, the number of black pearls (processors) is split evenly in a set, and each set contains at most two strings (consecutive leaves from the decomposition tree T). Thus, at level $\lceil \lg n \rceil$, each set (leaf of T') contains at most one processor.

It remains to prove the bound on the rates w'_i of information transfer in and out of each subtree of the balanced decomposition tree T'. Each subtree of T' corresponds to at most two strings of leaves from the original decomposition tree T. From Lemma 7, these two strings correspond to a forest of complete binary trees with at most four trees of a given height. All external communication of a complete binary subtree of a decomposition tree occurs through the surface corresponding to its root. Thus, the external communication per unit time of a subtree of T' is bounded by the sum of bandwidths from the roots of the corresponding complete binary subtrees of T. ∎

Corollary 9: Let R be a routing network that has a (w, a) decomposition tree for $1 < \alpha \leq 2$. Then S has a $(4(\alpha/\alpha - 1)w, \alpha)$ balanced decomposition tree.

Proof: The summation in Theorem 8 becomes a geometric series. ∎

VI. Universality of Fat-Trees

We now show that a fat-tree is universal for the amount of interconnection hardware it requires in the sense that any other routing network of the same volume can be efficiently simulated. From a theoretical point of view, we define "efficiently" as meaning at most polylogarithmic slowdown. Polylogarithmic time in parallel computation corresponds to polynomial time for sequential computation.

Some may argue that polylogarithmic slowdown may not be efficient if the exponent of the logarithm is large. The ability of one parallel computer to simulate another, however, merely gives confidence in the general-purpose nature of the computer. The loss of efficiency in the simulation is not felt if the parallel computer is programmed directly.

Many of the networks currently being built are not universal (for example, two-dimensional arrays, simple trees, or multigrids). These networks exhibit polynomial slowdown when simulating other networks. Thus, they have no theoretical advantage over a sequential computer which can easily simulate a network with polynomial slowdown. Interestingly, hypercube-based networks are universal for volume $\Theta(n^{3/2})$, but as we have observed, they do not scale down to smaller volumes.

Theorem 10: Let FT be a universal fat-tree on a set of n processors that occupies a cube of volume v, and let R be an arbitrary routing network on a set of n processors that also occupies a cube of volume v. Then there is an identification *of the processors in FT with the processors of R with the following property. Any message set M that can be delivered in time t by R can be delivered by FT (off-line) in time $O(t \lg^3 n)$.*

Proof: By Theorem 5, the routing network R has an $(O(v^{2/3}), \sqrt[3]{4})$ decomposition tree, and hence by Corollary 9, it also has an $(O(v^{2/3}), \sqrt[3]{4})$ balanced decomposition tree. Identify the processors at the leaves of the balanced decomposition tree of R in the natural way, with the processors at the leaves of the fat-tree FT.

By assumption, routing network R can deliver all the messages in a message set M in time t. In unit time, at most $O(v^{2/3}/2^{2k/3})$ messages can enter or leave a subtree rooted at level k in R's balanced decomposition tree. Thus, in t time, the total number of messages that can enter or leave a subtree rooted at level k is $O(tv^{2/3}/2^{2k/3})$.

There is a second bound on the transfer of information in and out of a subtree of the balanced decomposition tree of R. The number of messages that can enter or leave a single processor in time t is $O(t)$ since the number of connections to a processor is constant. Since there are at most $n/2^k$ processors in a subtree rooted at level k, the total number of messages that can enter or leave this subtree in t time is $O(tn/2^k)$.

We now compute an upper bound on the load factor $\lambda(M)$ that M puts on the fat-tree FT. Let c be a channel at level k in FT. We have just seen that the number load(M, c) of messages of M that must go through c is $O(tv^{2/3}/2^{2k/3})$ and $O(tn/2^k)$. Since FT is a universal fat-tree with root capacity $\Theta(v^{2/3}/\lg(n/v^{2/3}))$, the capacity of c is

$$\operatorname{cap}(c) = \min\left\{ \left\lceil \frac{n}{2^k} \right\rceil, \Theta\left(\frac{v^{2/3}}{2^{2k/3} \lg(n/v^{2/3})} \right) \right\}.$$

Thus, the load factor on c due to M is

$$\lambda(M, c) = O(t \lg(n/v^{2/3})),$$

and the load factor on the whole fat-tree is

$$\lambda(M) = O(t \lg(n/v^{2/3})).$$

The off-line routing result from Theorem 1 says that $O(t \lg(n/v^{2/3}) \lg n)$ delivery cycles are sufficient to route all the messages in M. Since the fat-tree can execute an off-line delivery cycle in $O(\lg n)$ time, the result follows. ■

The $O(\lg^3 n)$ factor lost in simulation is attributable to the channel capacities, the routing algorithm, and the switching. Of these three, only the last, the $O(\lg n)$ switching time for a delivery cycle, seems to be a necessary cost.

The first $O(\lg n)$ factor (actually $O(\lg(n/v^{2/3}))$) is because a fat-tree of volume v has a root capacity of $O(v^{2/3}/\lg(n/v^{2/3}))$. This logarithmic factor vanishes for the simulation of networks that have only slightly less $(O(v/\lg^{3/2}(n/v^{2/3})))$ volume. We have chosen to put all the simulation expense in time so that the comparison will be equal hardware versus equal hardware.

The second $O(\lg n)$ factor is lost by the off-line routing algorithm. In fact, we have recently discovered [8] that off-line routing in $O(\lambda(M) + \lg n \lg \lg n)$ delivery cycles is always possible. Moreover, if we assume that each processor has $\Theta(\lg n)$ connections, as is required by a Boolean hypercube, and each channel has capacity $\Omega(\lg n)$, Corollary 2 from Section III allows us to route in $O(\lambda(M))$ delivery cycles.

An important application of the universality of fat-trees is to the simulation of *fixed-connection networks*, that is, networks that have direct connections between processors. Here we relax the technical assumption in the definition of a universal fat-tree to allow the processors to have a given number d of connections to the routing network, instead of 1. Such a universal fat-tree of volume $O(v \lg^{3/2}(n/v^{2/3}))$ on n processors can simulate an arbitrary degree d fixed-connection network of volume v on n processors with only $O(\lg n)$ time degradation. The idea is that the channel capacities of the universal fat-tree are sufficiently large that the connections implied by the network can be represented as a one-cycle message set, which requires $O(\lg n)$ time to be delivered.

High-volume universal fat-trees can be compared to classical permutation networks, which all require $\Omega(n^{3/2})$ volume. A universal fat-tree on n processors with $O(n^{3/2})$ volume can route an arbitrary permutation off-line in time $O(\lg n)$. Up to constant factors, this is the best possible bound (assuming bounded-degree processors), but it is also achievable, for instance, by Benes networks [2], [34] or by on-line sorting networks [1], [15].

A natural extension to the off-line routing results presented here, and indeed, the one that motivates the entire paper, is the problem of on-line routing in fat-trees. Not surprisingly, there are universal fat-trees for on-line routing. In results to be reported elsewhere [8] we have discovered a randomized routing algorithm that delivers all messages in $O(\lambda(M) + \lg n \lg \lg n)$ delivery cycles with high probability [8], but the nodes of the fat-tree have somewhat different structure from the design given here. Using this result and essentially the construction given in this paper, one can obtain an on-line analog to Theorem 10, except with an $O(\lg^3 n \lg \lg n)$ time degradation. We anticipate further research will improve this bound.

VII. Concluding Remarks

Universality has been studied more generally in the parallel computation literature. Valiant [33] and Valiant and Brebner [31] have discovered universal routing schemes for large-volume networks. Galil and Paul [7] have proposed a general-purpose parallel processor based on the cube-connected-cycles network [25] that can simulate any other parallel processor with only a logarithmic loss in efficiency. Valiant [30] has shown that there are classes of universal Boolean circuits. A universal circuit of a given size can be programmed to simulate any circuit whose size is only slightly smaller. Fiat and Shamir [6] have proposed a universal architecture for systolic array interconnections.

Universal fat-trees are parameterized not only in the number of processors, but also in volume, which is indirectly a measure of communication potential. By considering arbitrary networks in terms of these two parameters, we have seen that the one fat-tree architecture is near-optimal throughout the entire range of the parameters. For communication-limited engineering situations, one need not necessarily retreat to special-purpose devices in order to compute efficiently in parallel.

Fat-trees have the advantage that they are a robust engineering structure. In principle, one need not worry about the exact capacities of channels as long as the capacities exhibit reasonable growth as we go up the tree. As a practical matter, one should build the biggest fat-tree that one can afford, and the architecture automatically ensures that communication bandwidth is effectively utilized. Another feature of fat-trees is that algorithms are the same no matter how big the fat-tree is. Code is portable in that it can be moved between an inexpensive computer and a more expensive one. Finally, the root channel offers a natural high-bandwidth external connection.

Although universal fat-trees have many desirable properties, there are many issues in the design of a routing network that we have not faced directly. For example, despite our concern for wirability, we have not presented a practical packaging scheme. Possibly, the packaging techniques for trees from [4] and [18] can be exploited. The constraints to be faced in packaging, however, will only be more stringent than the surface area constraint given in Assumption L3. We have attempted to deal with "pin boundedness" in a simple mathematical model, and our results should generalize to more complicated packaging models.

Another issue that we have not addressed is how messages should be sent in the network. The choice of the bit-serial approach in Section II has the advantage that the hardware is cheaper, but we may be paying in the performance of the routing algorithm. We also assumed the architecture was synchronized by delivery cycle. Presumably, fat-tree architectures can be built with different design decisions.

Whether the notion of universal parallel supercomputers is consistent with engineering reality, however, remains an open question. Independent of routing network issues, there are many other problems that must be solved if abstract n-processor parallel supercomputers are to become a reality. For example, problems of maintenance, fault tolerance, clock distribution, and reliable power supply must be solved. The hardware mechanisms needed for synchronization and instruction distribution, which are simple for single-processor machines, may be sufficiently complicated to overwhelm the advantages of having many processors.

But the largest problem that must be solved in parallel supercomputing seems to us to be the problem of programming the system with the concerns of both programming abstraction and algorithmic integrity (computational resources are not free). A supercomputer should not be a mere supercalculator (good at one restricted algorithm). It should have the power to efficiently execute many different parallel algorithms and to easily combine the results of separate parallel computations. A universal machine has the power, not just of any other machine, but of all other machines.

ACKNOWLEDGMENT

T. Knight of the M.I.T. Artificial Intelligence Laboratory inspired the author with many discussions about connection-machine architectures, which led him towards the ideas in this paper. He is to blame for the name "fat-tree." Thanks to R. Greenberg of the M.I.T. Laboratory for Computer Science for finding many bugs in various versions of the paper. Thanks also to A. Bawden, T. Bui, B. Chor, T. Cormen, O. Goldreich, A. Ishii, T. Leighton, B. Maggs, M. Maley, G. Miller, C. Phillips, R. Rivest, and P. Shor for their helpful comments and technical assistance.

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, "Sorting in c log n parallel steps," *Combinatorica*, vol. 3, no. 1, pp. 1–19, 1983.

[2] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965.

[3] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problems," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 300–343, Apr. 1984.

[4] S. N. Bhatt and C. E. Leiserson, "How to assemble tree machines," in *Advances in Computing Research, Vol. 2*. Greenwich, CT: Jai Press, 1984, pp. 95–114.

[5] S. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge, MA: M.I.T. Press, 1979.

[6] A. Fiat and A. Shamir, "Polymorphic arrays: A novel VLSI layout for systolic computers," in *Proc. IEEE 25th Annu. Symp. Foundations Comput. Sci.*, Oct. 1984, pp. 37–45.

[7] Z. Galil and W. J. Paul, "An efficient general-purpose parallel computer," *J. ACM*, vol. 30, no. 2, pp. 360–387, Apr. 1983.

[8] R. I. Greenberg and C. E. Leiserson, "Randomized routing on fat-trees," in *Proc. IEEE 26th Annu. Symp. Foundations Comput. Sci.*, Nov. 1985, to appear.

[9] W. D. Hillis, "The connection machine," Artif. Intell. Lab., Mass. Inst. Technol., Tech. Memo. 646, Sept. 1981.

[10] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. Vazirani, and V. Vazirani, "Global wire routing in two-dimensional arrays," in *Proc. IEEE 24th Annu. Symp. Foundations Comput. Sci.*, Nov. 1983, pp. 453–459.

[11] T. F. Knight, "The cross-omega router," Mass. Inst. Technol., Cambridge, unpublished manuscript, 1984.

[12] F. T. Leighton, "New lower bound techniques for VLSI," *Math. Syst. Theory*, vol. 17, no. 1, pp. 47–70, Apr. 1984.

[13] ——, "A layout strategy for VLSI which is provably good," in *Proc. 14th Annu. ACM Symp. Theory Comput.*, May 1982, pp. 85–98.

[14] ——, *Complexity Issues in VLSI*. Cambridge, MA: M.I.T. Press, 1983.

[15] ——, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, vol. C-34, pp. 344–354, Apr. 1985.

[16] F. T. Leighton and A. L. Rosenberg, "Three dimensional circuit layouts," Lab. Comput. Sci., Mass. Inst. Technol., Tech. Rep. MIT-LCS-TM-262, June 1984.

[17] C. E. Leiserson, "Area-efficient graph layouts (for VLSI)," in *Proc. IEEE 21st Annu. Symp. Foundations Comput. Sci.*, Oct. 1980, pp. 270–281.

[18] ——, *Area-Efficient VLSI Computation*. Cambridge, MA: M.I.T. Press, 1983.

[19] R. J. Lipton and R. E. Tarjan, "A planar separator theorem," *SIAM J. Appl. Math.*, vol. 36, no. 2, pp. 177–189, Apr. 1979.

[20] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[21] M. S. Pinsker, "On the complexity of a concentrator," in *Proc. 7th Int. Teletraffic Conf.*, Stockholm, Sweden, June 1983, pp. 318/1–318/4.

[22] N. J. Pippenger, "The complexity theory of switching networks," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, Aug. 1973.

[23] ——, "Superconcentrators," *SIAM J. Comput.*, vol. 6, no. 2, pp. 298–304, June 1977.

[24] ——, "Parallel communication with limited buffers," in *Proc. IEEE 25th Annu. Symp. Foundations Comput. Sci.*, Oct. 1984, pp. 127–136.

[25] F. P. Preparata and J. E. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, vol. 24, no. 5, pp. 300–309, May 1981.

[26] A. L. Rosenberg, "Three-dimensional integrated circuitry," in *Proc. CMU Conf. VLSI Syst. Comput.*, H. T. Kung, R. Sproull, and G. Steele, Eds., Oct. 1981, pp. 69–79.

[27] J. T. Schwartz, "Ultracomputers," *ACM Trans. Programming Lang. Syst.*, vol. 2, no. 4, pp. 484–521, 1980.

[28] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[29] C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[30] L. G. Valiant, "Universal circuits," in *Proc. 8th Annu. ACM Symp. Theory Comput.*, May 1976, pp. 196–203.

[31] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, May 1981, pp. 263–277.

[32] L. G. Valiant, "Universality considerations in VLSI circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 135–140, Feb. 1981.

[33] ——, "A scheme for fast parallel communication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350–361, May 1982.

[34] A. Waksman, "A permutation network," *J. ACM*, vol. 15, no. 1, pp. 159–163, Jan. 1968.

Charles E. Leiserson (M'83) received the B.S. degree in computer science and mathematics from Yale University, New Haven, CT, in 1975 and the Ph.D. degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1981.

He is currently an Associate Professor of Computer Science and Engineering at the Massachusetts Institute of Technology (M.I.T.), Cambridge. In 1981 he joined the faculty of the Theory of Computation Group in the M.I.T. Laboratory for Computer Science. His expertise includes parallel computation, VLSI architectures, graph theory, digital circuit timing, analysis of algorithms, computer-aided design, placement and routing, wafer-scale integration, layout compaction, and most recently, parallel supercomputing. His principal interest is in the theoretical foundation of parallel computation, especially as it relates to engineering reality.

Prof. Leiserson has authored over 20 papers on the theory of VLSI and parallel algorithms. As a graduate student at Carnegie-Mellon he wrote the first paper on systolic architectures with H. T. Kung, for which they received a U.S. patent. His Ph.D. dissertation, "Area-Efficient VLSI Computation," which deals with the design of systolic systems and with the problem of determining the VLSI area of a graph, won the first ACM Doctoral Dissertation Award. He was awarded a Presidential Young Investigator Award in 1985. He is a member of the ACM, and he serves on the ACM General Technical Achievement Award Committee, which selects the Turing Award winner.