Deterministic On-Line Routing on Area-Universal Networks*

Paul Bay**
Gianfranco Bilardi***

TR 91-1237
September 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported in part by the National Science Foundation under grant MIP-86-02256 and by the Joint Services Electronics Program under contract F49620-87-C-0044. An extended abstract of this paper was published in *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (St. Louis, MO, Oct. 22-24). IEEE Computer Society Press, Los Alamitos, CA 1990, pp. 297-306.
**Computer Science Department, Cornell University, Ithaca, NY 14853.
***Computer Science Department, Cornell University and Dipartimento di Elettronica ed Informatica, Universita' di Padova, Via Grandenigo 6/A, 35131 Padova, Italy.

Deterministic On-Line Routing on Area-Universal Networks*

Paul Bay† and Gianfranco Bilardi‡

September 23, 1991

Abstract

Two deterministic routing networks are presented: the *pruned butterfly* and the *sorting fat-tree*. Both networks are area-universal, i.e., they can simulate any other routing network fitting in similar area with polylogarithmic slowdown. Previous area-universal networks were either for the off-line problem, where the message set to be routed is known in advance and substantial precomputation is permitted, or involved randomization, yielding results that hold only with high probability. The two networks introduced here are the first that are simultaneously deterministic and on-line, and they use two substantially different routing techniques. The performance of their routing algorithms depends on the difficulty of the problem instance, which is measured by a quantity λ known as the load factor. The pruned butterfly algorithm runs in time $O(\lambda \log^2 N)$, where N is the number of possible sources and destinations for messages and λ is assumed to be polynomial in N. The sorting fat-tree algorithm runs in $O(\lambda \log N + \log^2 N)$ time for a restricted class of message sets including partial permutations. Other results of this work include a new type of sorting circuit and an area-time lower bound for routers.

Categories and Subject Descriptors: C.1.2–Interconnection Architectures, C.2.1–Network Communications, C.2.1–Network Topology, F.1.2–Parallelism and Concurrency, F.2.2–Routing and layout, F.2.2–Computations on discrete structures, F.2.3

General terms: Algorithms, Theory

Additional Key Words and Phrases: fat-tree, area-universal, general purpose

1 Introduction

The performance of a general-purpose parallel computer depends fundamentally on the ability of the interconnection network to route arbitrary sets of messages quickly. Considerable attention has been given to the design of sparse interconnection networks and routing algorithms for them [VB81,

*This work was supported in part by the National Science Foundation under grant MIP-86-02256 and by the Joint Services Electronics Program under contract F49620-87-C-0044. An extended abstract of this paper was published in *Proceedings of the* 31^{st} *Annual Symposium on Foundations of Computer Science* (St. Louis, MO, Oct 22-24). IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 297–306.

†Computer Science Department, Cornell University, Ithaca, NY 14853.

‡Computer Science Department, Cornell University and Dipartimento di Elettronica ed Informatica, Universita' di Padova, Via Gradenigo 6/A, 35131 Padova, Italy.

Ale82, Upf84, Val82, Pip84, Ran87, LMR88]. If the cost of the network is taken to be the number of nodes, networks constructed using expander graphs generally achieve the best performance for routing and related operations [AKS83, Lei85b, PU87, PU89, HB88, Her89, Upf89, LM89, Her91]. The basic advantage of expanders is the high bandwidth available across most cuts of the network, although designing a routing algorithm that fully exploits this bandwidth often requires considerable ingenuity.

In the more realistic VLSI model [Tho80] we will adopt for this paper, the cost of a network is the chip area it occupies in a two-dimensional layout.[1] When the cost of the wires is taken into account, the problem of designing routing networks with the best performance has a different solution. In the VLSI model, high bandwidth networks like expanders give good routing performance only for the special case of when the number of terminals of the network is much smaller than the area cost. (A *terminal* of a routing network is a processor interface point that can be the source or destination of messages.) In a fundamental paper [Lei85b], Leiserson initiated the investigation of routing networks that are *area-universal*: they can route almost as efficiently as *any* other network of similar area, regardless of the number of nodes.

Another contribution of [Lei85b] was to relate a network's routing performance to a guarantee about its efficiency in simulating an arbitrary network of similar cost. Such a relation is made possible in part by expressing a routing algorithm's performance in terms of *load factor*, a measure of the difficulty of the routing problem instance. For a given network, the load factor λ of a set of messages is the maximum ratio of the number of messages that must cross a cut of the network to the number of wires that cross the cut. (Section 2 gives a more rigorous definition.) Leiserson devised an algorithm that runs in time $O(\lambda \log^2 N)$ time on a routing network with N terminals. His network has area $O(N \log^2 N)$ and can simulate an arbitrary N-terminal network of area $O(N \log^2 N)$ with a slowdown in time of only $O(\log^2 N)$.

[1] We discuss results only for two dimensions, but analogous three-dimensional results can be obtained in most cases with simple adaptations. When reporting a result formulated in the literature in three dimensions, we cite a two-dimensional equivalent.

The algorithm presented in [Lei85b] is for the *off-line* version of the routing problem. In off-line routing, the sources and destinations of the messages are known before the routing starts, and the network can be configured accordingly. The time to move messages from source to destination is the primary concern, rather than the time needed for configuring the network. Off-line routing has application when the pattern of communication is known at compile time. An interesting special case is when a known fixed-connection network is being simulated. In *on-line* routing, sources and destinations are known only at run time. Any general-purpose computer must have an efficient on-line routing algorithm so it can run programs that generate unpredictable, data-dependent message sets.

The first on-line routing algorithm for an area-universal network was proposed in [GL85, GL89]. There, an N-terminal network of area $O(N \log^2 N)$ uses randomization to route in time $O(\lambda \log^2 N \ \log\log N)$ with high probability. A faster randomized on-line routing algorithm, which permits en route combining of messages with the same destination, is offered in [LMR88], for a network with N terminals, $O(N \log^2 N)$ area, and $O(\lambda(M) \log N + \log^2 N)$ routing time with high probability[2]. In [Gre90], area-universality is investigated under alternative assumptions about wire delay, and area-universal networks with processors of various sizes are considered.

This paper gives the first *deterministic* solutions for area-universal on-line routing. We propose two routers, the *pruned butterfly* and the *sorting fat-tree*. The N-terminal pruned butterfly can route an arbitrary message set of polynomial size in time $O(\lambda \log^2 N)$ time and area $O(N \log^2 N)$.[3] The sorting fat-tree routes only message sets where at most a constant number of messages have their source or destination at any given terminal. However, on this important class of messages sets, the N-terminal sorting fat-tree has better performance: it takes area $O(N \log^2 N)$ and routes on-line in time $O(\lambda \log N + \log^2 N)$. For both networks, the routing time is within a factor of $O(\log N)$ of the $\Omega(\lambda \log N)$ bandwidth lower bound. For the special case of permutations with

[2] We have translated the results of [LMR88] from the word model to the bit model.

[3] A result similar to this one has been established independently, using a different construction, by Leiserson and Park (personal communication).

$\lambda = \Omega(\log N)$, the routing algorithm for the sorting fat-tree actually achieves the bandwidth lower bound. Note that this special case is common: for a random permutation, the expected value of λ is $\Omega(\sqrt{N})$. Both the pruned butterfly and the sorting fat-tree can simulate any N-terminal router of area $O(N \log^2 N)$ with a slowdown of $O(\log^2 N)$, or any router of area $O(N)$ with slowdown $O(\log N)$.

In the construction of the sorting fat-tree, we deploy a new type of sorting circuit of independent interest. It has area A and for any n with $\sqrt{A} \leq n \leq A/\log A$, it can sort n words of $(\log n + \Theta(\log n))$ bits with optimal $AT^2 = O(n^2 \log^2 n)$. Previously known VLSI circuits [BP85, Lei85a] achieve $AT^2 = O(n^2 \log^2 n)$, but for a fixed value of A, different circuits are needed for different values of n. The novel feature of our sorter is that the same circuit can process all input sizes in the given range in optimal time.

The rest of this paper is organized as follows: Section 2 defines our routing model and clarifies the relationship between routing performance and area-universality. Sections 3 and 4 present the pruned butterfly and the sorting fat-tree, respectively. Section 5 establishes an AT^2 lower bound for networks whose routing algorithms are not allowed to modify the content of messages (e.g., by recoding a pair of messages into another pair).

2 Preliminaries

We model a routing network as a graph whose vertices represent constant-size logic gates, and whose edges represent wires connecting the corresponding gates. The networks we present are synchronous, that is, some global timing mechanism permits topologically distant gates to switch simultaneously. A distinguished subset of N vertices called *terminals* perform I/O functions and can be sources or destinations of messages. A *message* is a triple $\langle \mathit{src}, \mathit{dst}, \mathit{body} \rangle$, where *src* and *dst* are $\log N$-bit terminal id's and *body* is a b-bit string. (Typically we will assume $b = \Theta(\log N)$.) The messages of the input set M are initially made available at their source terminals, and the task of the router is to deliver the message bodies to the destinations. The *degree* of a message set, $d(M)$, is the maximum number of messages that have their source or destination at a given terminal. If $d(M) = 1$, M is called a *partial permutation.* Sometimes we will find it useful to think of a set of messages M as a graph with one vertex for each of the N terminals and an edge between vertices u and v if M contains a message with source u and destination v.

The time to route a message set M on an N-terminal routing network is stated in terms of a parameter $\lambda(M)$, known as the *load factor* of the message set. If U is a subset of the terminals, a *cut* S is a set of edges such that every path from a terminal in U to a terminal in $\overline{U}$ includes some edge in S. The *load* placed on cut S by M is defined as the number of messages that must cross it, and denoted $\ell(M, S)$. The *load factor* of a cut is $\ell(M,S)/|S|$, denoted $\lambda(M, S)$. The load factor on the entire network, $\lambda(M)$, is the maximum load factor of any cut. Clearly $b\lambda(M)$ is a lower bound on the number of bit steps necessary to route M.

Expressing a routing algorithm's performance in terms of the load factor measures how close to optimal it is for the network, but says nothing about how well the network can simulate other networks of similar area. To prove a network is area-universal it is helpful to have a measure of routing difficulty that reflects time constraints on routing that come from the network's layout. Consider any N-terminal router laid out in area $O(N)$. Any rectangle of area $\Theta(N/2^i)$ in a hierarchical decomposition of the layout would contain $\Theta(N/2^i)$ terminals enclosed by a perimeter of length $\Theta(\sqrt{N/2^i})$. The following definition captures the lower bound on routing time imposed by the limited bandwidth crossing the perimeter of any such rectangle.

Let T be an N-leaf complete binary tree with the leaves labeled 0 to $N-1$ from left to right. Let an edge at height h above the leaves have a weight of $2^{\lceil h/2 \rceil}$. Now consider the natural embedding of the message set's graph in T, where each message edge is mapped to the simple path in the tree from the leaf labeled by its source to the leaf labeled by its destination. Then $\eta(M)$, the *reference load factor*, is defined to be the maximum over all edges e of T of the congestion of e divided by its weight.

The usefulness of the reference load factor stems from the following result, a simple variant of

Theorem 10 in [Lei85b].

Proposition 1 *If the terminals of an N-terminal router can be labeled in such a way that it can deliver any message set M in time $O(b\eta(M)\tau(N))$, then it can simulate any N-terminal router of area A, losing at most a factor of $O(\sqrt{A/N}\tau(N))$ in time.*

In this paper and in the rest of the literature, Proposition 1 is used to prove universality results in two steps. The first is to design an N-terminal router with area not much larger than N in such a way that, for any message set M, $\lambda(M) \leq \eta(M)$. The second is to design a routing algorithm for the network that runs in time close to $O(\lambda(M))$. The first step was made easier by the introduction in [Lei85b] of the "fat-tree" framework for routing networks. A fat-tree is a complete binary tree with subnetworks at the nodes that perform switching functions. Two neighboring nodes in the tree are joined by a group of wires called a channel. The number of edges in a channel c is referred to as its *capacity*, written $\mathrm{cap}(c)$. The fat-trees proposed in the literature and in this paper have the useful property that the cuts corresponding to the channels are sufficient for determining the load factor. Thus choosing the capacity of a channel at height h above the leaves to be $2^{\lceil h/2 \rceil}$ guarantees $\lambda(M) \leq \eta(M)$.

3 Deterministic On-line Routing on the Pruned Butterfly

In this section we present the pruned butterfly switching network (Subsection 3.1) which, augmented with some auxiliary circuitry (Subsection 3.2), supports an efficient on-line routing algorithm for arbitrary message sets (Subsection 3.3).

3.1 The Pruned Butterfly

We give the name *pruned butterfly* to the graph $G(V, E)$ defined as follows, for N a power of 4.

$$V = \{\langle i, j, k\rangle : 0 \leq i \leq \log N,\ 0 \leq j < 2^i,\ 0 \leq k < \sqrt{N}2^{-\lfloor i/2 \rfloor}\},$$

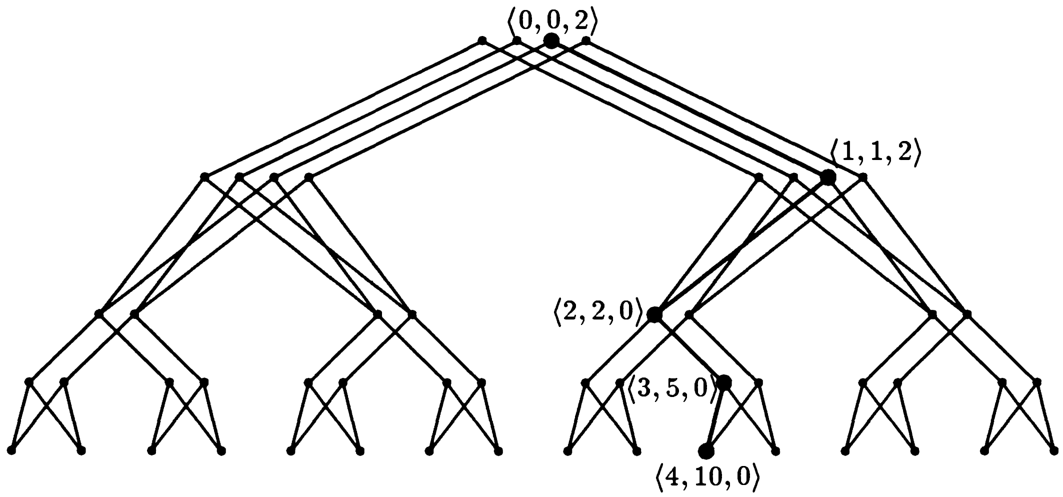

Figure 1: The pruned butterfly when $N = 16$. The triples identify the vertices on the unique path from leaf 10 to root 2.

$$E = \bigcup_{i=1}^{\log N} \bigcup_{j=0}^{2^i-1} E_{ij},$$

where, for even i,

$$E_{ij} = \{(\langle i,j,k\rangle, \langle i-1, \lfloor j/2 \rfloor, k\rangle),\ (\langle i,j,k\rangle, \langle i-1, \lfloor j/2 \rfloor, k + \sqrt{N}2^{\lfloor -i/2 \rfloor}\rangle) :\ 0 \le k < \sqrt{N}2^{-\lfloor i/2 \rfloor}\},$$

and for odd i,

$$E_{ij} = \{(\langle i,j,k\rangle, \langle i-1, \lfloor j/2 \rfloor, k\rangle) :\ 0 \le k < \sqrt{N}2^{-\lfloor i/2 \rfloor}\}.$$

As illustrated in Figure 1, the pruned butterfly has the structure of a fat-tree. We use the term *node* to refer to the set of pruned butterfly vertices that together form one vertex of the underlying complete binary tree. Node j in the left-to-right ordering of all fat-tree nodes at depth i is the set $\{\langle i,j,k\rangle | 0 \le k < \sqrt{N}2^{-\lfloor i/2 \rfloor}\}$. The edges in E_{ij} form a channel of capacity $|E_{ij}| = \sqrt{N}2^{-\lfloor (i-1)/2 \rfloor}$, between a level i node and its parent.

As shown in Figure 2, the pruned butterfly is a subgraph of the butterfly. A similar graph has been used in [LMR88] for randomized routing.

Vertex $\langle \log N, l, 0\rangle$ of the pruned butterfly is called *leaf* l and vertex $\langle 0,0,r\rangle$ is called *root* r. Between leaf l and root r is a unique shortest path that includes exactly one vertex from each level

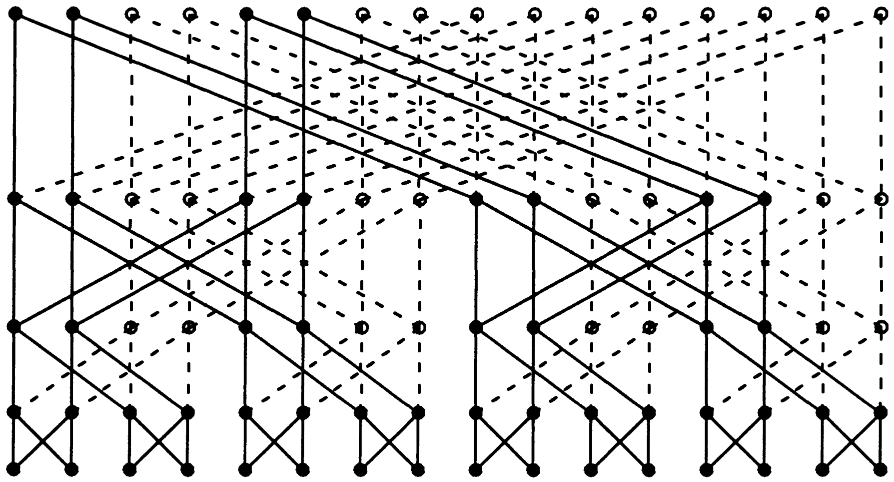

Figure 2: The pruned butterfly is a subgraph of the butterfly.

of the tree. If l and r have binary representations $\alpha_{\log N-1}\cdots\alpha_0$ and $\beta_{(\log N)/2-1}\cdots\beta_0$, respectively, then the level i vertex of the path is $\langle i, \lfloor l/2^{\log N-i}\rfloor, r \bmod 2^{\lceil(\log N-i)/2\rceil}\rangle$ or, with slight abuse of notation, $\langle i, \alpha_{\log N-1}\cdots\alpha_{\log N-i}, \beta_{\lceil(\log N-i)/2\rceil-1}\cdots\beta_0\rangle$.

We now regard the vertices of the pruned butterfly graph as switches and the edges as wires, and consider movement of messages in the resulting switching network. A set of $s \le \sqrt{N}$ messages $m_0, m_1, \ldots, m_{s-1}$ is called an *expansion* if m_h has source at root h and destination at leaf l_h, and $l_{h-1} \le l_h$ for $h = 1, \ldots, s-1$. A similar message set is called a *compression* if m_h has source at leaf l_h and destination at root h, and $l_{h-1} \le l_h$ for $h = 1, \ldots, s-1$.

Let us consider the following *greedy* routing strategy. Messages may be active or inactive; initially all messages are active. At each step, each active message tries to advance one level on its unique source-to-destination path. Two messages *conflict* at an edge if their paths both contain the edge. A message m_h becomes inactive and advances no further if it conflicts with a message m_k where $k < h$. This simple strategy has the following useful property.

Lemma 1 *If the greedy strategy is applied to an expansion and a conflict arises at an edge of channel c, then all the edges of c are occupied by active messages.*

Proof: We will establish the following property inductively: the messages arriving at any fat-tree node have consecutive sources. The property is trivially true at the root. Suppose the property holds for all nodes at depth $i-1$ or less, and consider a level $i-1$ node u joined to its child v by channel c. Let M_v be the subset of messages arriving at u whose destinations are in v's subtree. From the induction hypothesis, the messages of M_v have consecutive sources, so if there is no conflict at any edge of c, all of M_v remains active and arrives at v.

Now suppose there is a conflict at some edge of c. Since the messages of M_v have consecutive sources, we conclude from the structure of their unique paths to u that they arrive at vertices of u that are consecutive modulo $\text{cap}(c)$. Therefore a conflict can occur only between two messages whose sources differ by $\text{cap}(c)$. In this case the conflict resolution rule guarantees that the $\text{cap}(c)$ messages in M_v with the smallest destinations remain active and arrive at v. Thus not only do the messages arriving at v have consecutive sources, establishing the induction hypothesis, but every edge of c is occupied, establishing the lemma. □

Corollary 1 *If the load factor of an expansion does not exceed 1, the expansion is routed by the greedy strategy without conflicts. By symmetry, the same property holds for a compression.*

An interesting property of the N-leaf pruned butterfly (reportedly [GL89] already observed by Leiserson and Leighton) is that it embeds an N-leaf mesh of trees with constant dilation and load. Therefore, the $\Omega(N \log^2 N)$ lower bound for the area of the mesh of trees [Lei84] transfers to the pruned butterfly. An $O(N \log^2 N)$ layout of the latter graph is easily achieved by the H-tree method, as shown in Figure 3. Moreover, mesh of trees algorithms can be readily adapted to the pruned butterfly.

3.2 Auxiliary Circuitry

For the on-line routing of a message set, the switching structure of the pruned butterfly needs to be augmented with some circuitry supporting auxiliary functions such as buffering, counting, sorting,

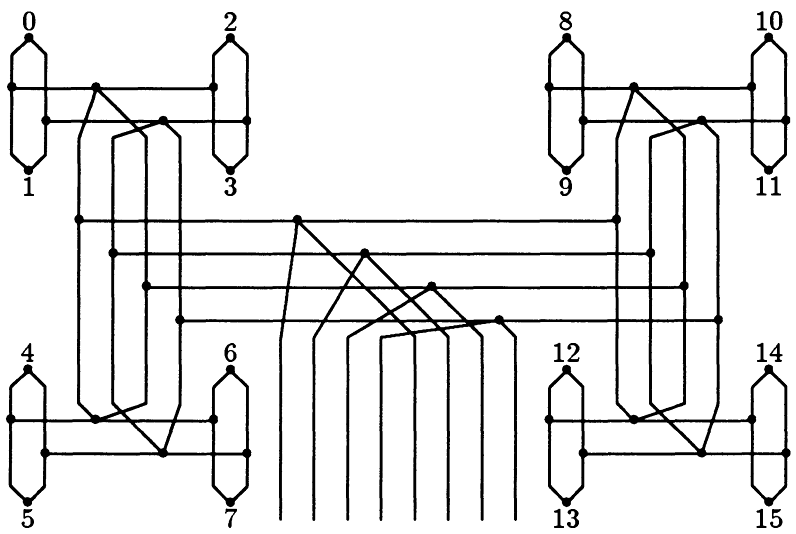

Figure 3: The H-tree layout of a 16-leaf subtree of the pruned butterfly.

and partial-sum computation.

Each leaf node of the pruned butterfly contains a terminal, a constant-area, bit-serial interface with a processor. The processor stores a set of messages, each consisting of a record with an $O(\log N)$-bit information field, a $\log N$-bit destination field, and a $\lceil \log\log N \rceil$-bit peak-level field. (A message's *peak* is the level in the tree of the lowest common ancestor of the message's current position and destination.) The processor keeps the messages in a priority queue organized by peak-level (minimum level at the top) which can receive—every $O(\log N)$ bit steps—either an insert or a deletemin instruction from the leaf.

A leaf node is responsible for initializing the peak-level of the messages originating at the attached processor, and for updating the peak-level of a message before inserting it into the queue. A leaf also maintains $\log N$ counters, each storing the number of messages in the queue with a given peak level. In addition a leaf is equipped with a comparator for $\log N$-bit numbers and a circuit to compute $a \bmod b$ where a and b are $O(\log N)$-bit numbers. The leaf can perform any of the operations mentioned in $O(\log N)$ time, and can be laid out in a square region of side length $O(\log N)$. The leaf's area limits the length of the peak-level counters to $O(\log N)$, which in turn limits the network to handling message sets with cardinality polynomial in N.

By adding a single-bit full adder and an $O(\log N)$-bit shift register to each node of the pruned butterfly tree, and using a straightforward bit-pipelined version of the tree implementation of prefix computation algorithms [DS82, BP89], we have:

Lemma 2 *Let T be an n-leaf subtree of the pruned butterfly routing network where a register in leaf i stores an integer $0 \leq b_i < N$. In $O(\log N)$ time T can compute and store at leaf i (a) the partial sum $\sum_{j=0}^{i-1} b_j$ or (b) the total sum $\sum_{j=0}^{n-1} b_j$.*

By adding $\sqrt{n}$ $O(\log N)$-bit buffers to the root node of every n-leaf subtree T, and adapting the mesh of trees sorting algorithm [NMB83, Lei81] to the pruned butterfly, we have:

Lemma 3 *Let T be an n-leaf subtree of the pruned butterfly routing network, and suppose a message set M of cardinality at most $\sqrt{n}$ is initially at the root of T. Then in $O(\log N)$ time the message set can be sorted by destination and output at the root T.*

The node at the root of an n-leaf subtree can be laid out in area $O(n + \log N)$, and a leaf can be laid out in area $O(\log^2 N)$, so an H-tree layout of the entire circuit takes area $\Theta(N \log^2 N)$.

3.3 Routing Algorithm

Routing of a message set M is performed in $\log N$ stages: stage $0, \ldots,$ stage $\log N - 1$. Let M_i be the set of messages with peak at level i at the beginning of stage i. During stage i each message of M_i is moved to a new leaf, possibly different from the final destination, but always lowering the message's peak. A message not routed to its true destination is said to be *sidetracked* and is processed again in the stage associated with its new peak. A crucial property maintained by the algorithm is that, for each $i, \lambda(M_i)$ is $O(\lambda(M))$.

Stage i is conveniently described in terms of the activity of a generic subtree T with root at level $i+1$. Such a subtree interacts only with its sibling, T', to which it sends and from which it receives some messages. Let M_T be the set of messages in M_i with source in T, and let

$$\lambda_a(M_T) = \max_{c \in T} \ell(M_T, c)/\text{cap}(c)$$

be its *ascending load factor.* Stage i of the algorithm partitions M_T into $\lambda_T = \lceil \lambda_a(M_T) \rceil$ batches and routes each batch to leaves of T' in $O(\log N)$ time. Although some messages of M_T may not reach their true destinations, the routing policy guarantees that λ_T is $O(\lambda(M))$, so stage i takes $O(\lambda(M) \log N)$ time.

We now describe and analyze the algorithm for stage i of the routing.

Partition of M_T into batches. First, each subtree T rooted at level $i+1$ computes λ_T and makes this quantity available at each of its leaves, as follows. Each leaf loads the value of its peak-level counter for level i into a designated register. By a leaf-to-root summation of these values, each tree node obtains $\ell(M_T, c)$, where c is the channel connecting that node to its parent. The node then computes $\lceil \lambda(M_T, c) \rceil = \lceil \ell(M_T, c)/\text{cap}(c) \rceil$. (As cap$(c)$ is a power of two, this amounts to a right-shift with truncation.) Another leaf-to-root computation makes $\lambda_T = \max_{c \in T} \lceil \lambda(M_T, c) \rceil$ available at the root of T, which can then broadcast it to all the leaves.

Second, for each leaf l, T computes x_l, the number of messages of M_T with source to the left of l. (This is accomplished by applying part (a) of Lemma 2 to the leaves' peak-level counters for level i.) If l contains y_l messages of M_T, then it contributes a message to batch $B_{(x_l+x) \bmod \lambda_T}$, for each x such that $0 \leq x < y_l$. It is easy to see that for each batch B so defined, $\lceil \lambda_a(B) \rceil = 1$.

Thus far the algorithm involves a constant number of $O(\log N)$ time operations. Therefore we have:

Lemma 4 *In $O(\log N)$ time, M_T can be partitioned into λ_T batches, $B_0, \ldots, B_{\lambda_T - 1}$, such that $\lambda_a(B_0), \ldots, \lambda_a(B_{\lambda_T-1}) \leq 1$.*

Routing of the batches. The messages in a batch B are first routed to the root of T by a compression operation. Specifically, by a prefix computation, each leaf of T containing a message m in B determines the number h of messages of B to its left ($0 \leq h <$ number of vertices at the root of T). Then message m is routed to vertex $\langle i+1, j, h\rangle$ at the root of T. Since $\lambda_a(B) \le 1$ (Lemma 4), the routing paths do not conflict (Corollary 1). The bits in the binary representation of h correspond directly to the binary choices message m encounters at even levels on its path to $\langle i+1, j, h\rangle$. Thus the routing can be done bit-serially in $O(\log N)$ time if the message packets are organized with the least significant bits of h at the front.

Once a batch B reaches the root of T, it is transferred to the root of T' and sorted by destination. By Lemma 3, this takes $O(\log N)$ time. Now B is an expansion and is routed to the leaves of T' by the following variant of the greedy strategy described earlier.

A bit is prepended to each message, indicating whether the message is active or sidetracked. The bit is initialized to "active". A *direction bit* is associated with each pruned butterfly vertex and initialized to "left". Routing is bit-serial. Decisions are decentralized and made on-the-fly by individual vertices of the pruned butterfly according to the following rules. (a) If there is no conflict, an active message is routed toward its destination. (b) If two active messages compete for the same edge, then the one with the larger destination is sidetracked and its initial bit is toggled. (c) An active message has precedence over a sidetracked one. (d) Each time a vertex receives a single sidetracked message from its parent(s), it sends it down the edge indicated by the direction bit of that vertex, and toggles the bit. (Notice that exactly the same set of active messages reaches each node as in the earlier greedy strategy; in particular, Lemma 1 applies to the present strategy as well.)

Assume that the message is organized in a packet whose first field is the destination address, the currently-most-significant bit of which is the first to arrive at a pruned butterfly vertex. Based on this bit, the vertex can implement the policy described above in constant time. If the message is active, then the bit is stripped away before the message goes to the next vertex. The only subtlety arises in the implementation of rule (b), because a vertex cannot determine which of two conflicting messages has the larger destination by comparing only the most significant address bits. However, the vertex can send the (identical) address bits down both edges, delaying the routing decision until a distinguishing bit appears.

Since messages have $O(\log N)$ bits and paths have $O(\log N)$ vertices, the bit-serial pipelined routing of a batch from the root to the leaves of T' takes $O(\log N)$ time. Thus, the entire routing of one batch takes $O(\log N)$ time and, as there are λ_T batches, we conclude:

Lemma 5 *All messages in M_T can be routed to leaves of T' in time $O(\lambda_T \log N)$.*

Our next goal is to bound λ_T in terms of $\lambda(M)$. Note that M_T can be partitioned into $M_T^s \cup M_T^o$, where M_T^s is the set of messages destined to T' that have been sidetracked to T during stages 0 through $i-1$, and M_T^o is the set of messages that have never been moved from their original sources at leaves of T. Clearly, $\lambda_a(M_T^o) \leq \lambda(M)$, and since $\lambda_a(M_T) \leq \lambda_a(M_T^s) + \lambda_a(M_T^o)$, all that remains is to bound $\lambda_a(M_T^s)$.

Let c' be the channel between the root of T' and its parent. By Lemma 1 applied to c', a batch contributes messages to M_T^s only if $\text{cap}(c)$ active messages from the same batch traverse c'. By definition of $\lambda(M, c')$, there can be at most $\lceil \lambda(M, c') \rceil \leq \lceil \lambda(M) \rceil$ such batches. Since each batch B is delivered to destinations that ensure $\lambda(B) \leq 1$, we conclude that $\lceil \lambda_a(M_T^s) \rceil \leq \lceil \lambda(M) \rceil$, and $\lambda_T = \lceil \lambda_a(M_T) \rceil \leq 2\lceil \lambda(M) \rceil$. In summary, we have:

Lemma 6 *For each subtree T, $\lambda_T \leq 2\lceil \lambda(M) \rceil$.*

Combining the above results, we arrive at the following theorem.

Theorem 1 *The pruned butterfly of N terminals can route a set M of $O(\log N)$-bit messages, with $|M|$ polynomial in N, and with load factor $\lambda(M)$, in time $O(\lambda(M) \log^2 N)$ and area $O(N \log^2 N)$.*

In the analysis of our routing algorithm, we have assumed that sidetracked messages reaching a given leaf of the pruned butterfly are temporarily stored in the priority queue of the attached processor. The leaf itself stores only the peak-level counters, as described in Section 3.2. We now want to bound the size of the processors' priority queues. While it is easily established that at most

$O(\lambda(M)\log N)$ messages (one per batch) can be sidetracked to the same leaf, a careful analysis yields a tighter bound in terms of $d(M)$, the message set's degree.

Lemma 7 *At no time during the routing of a message set of degree d does any processor store more than $(d+1)\log N + d$ sidetracked messages.*

Proof: Let T be a subtree of the pruned butterfly and let $|T|$ denote the number of its leaves. Choosing a time between batch routings when each message is at some leaf, we denote by $s(T)$ the total number of sidetracked messages stored at the leaves of T and by $x(T)$ the maximum number of sidetracked messages stored at a leaf of T. We shall show that the routing policy balances the storage load at the leaves so that, for any subtree T,

$$x(T) - s(T)/|T| \leq (d+1)\log|T|. \tag{1}$$

When T is the whole fat-tree, $|T| = N$ and $s(T) \leq dN$, so (1) yields the statement of the lemma.

We will establish (1) by induction on $|T|$. For $|T| = 1$, (1) is trivially satisfied, as both sides of the inequality are zero. For $|T| > 1$, let T_0 and T_1 denote the two subtrees of T so that $s(T_0) \leq s(T_1)$, and inductively assume that $x(T_h) - 2s(T_h)/|T| \leq (d+1)\log(|T|/2)$, for $h = 0, 1$. To prove (1) for T, observe that $x(T) = \max(x(T_0), x(T_1))$ and $s(T) = s(T_0) + s(T_1)$, and consider two cases.

Case $x(T_0) \geq x(T_1)$. Here $x(T) = x(T_0)$ and, since $s(T) \geq 2s(T_0)$,

$$x(T) - s(T)/|T| \leq x(T_0) - 2s(T_0)/|T| \leq (d+1)\log|T|,$$

where the last step uses the inductive hypothesis on T_0 and the relation $|T|/2 \leq |T|$.

Case $x(T_0) < x(T_1)$. Here $x(T) = x(T_1)$ and

$$x(T) - s(T)/|T| \leq x(T_1) - 2s(T_1)/|T| + (s(T_1) - s(T_0))/|T|. \tag{2}$$

We need to bound the quantity $s(T_1) - s(T_0)$. To this end, let m be a sidetracked message stored in T_h, for $h \in \{0,1\}$, and let v be the pruned butterfly vertex at the root of T from which m entered T_h. Message m can be one of four types. Either

1. m's final destination is a leaf of T_h, or if not, either

2. m arrived at v together with an active message destined to T_{1-h}, or

3. m arrived at v together with a sidetracked message routed into T_{1-h}, or

4. m arrived at v alone.

Clearly $s(T_h) = \sum_{a=1}^{4} s_a(T_h)$, where $s_a(T_h)$ is the number of messages in T_h of type a. Since at most $d|T|/2$ messages could have destinations in T_h, $s_1(T_1) - s_1(T_0) \leq s_1(T_1) \leq d|T|/2$ and $s_2(T_1) - s_2(T_0) \leq s_2(T_1) \leq d|T|/2$. By definition, $s_3(T_1) - s_3(T_0) = 0$. Finally, as single sidetracked messages arriving at v are sent alternately to T_h and T_{1-h}, we have that $s_4(T_1) - s_4(T_0) \leq \sqrt{2|T|}$, where $\sqrt{2|T|}$ is the maximum number of vertices at the root of T. Combining these four bounds yields

$$s(T_1) - s(T_0) \leq d|T| + \sqrt{2|T|}. \tag{3}$$

Substituting (3) into (2) and applying the inductive hypothesis on T_1 produces

$$\begin{aligned} x(T) - s(T)/|T| &\leq (d+1)\log(|T|/2) + d + \sqrt{2/|T|} \\ &\leq (d+1)\log|T| - d - 1 + d + \sqrt{2/|T|} \\ &\leq (d+1)\log|T|, \end{aligned}$$

where we have exploited the fact that, as $|T| \geq 2$, $\sqrt{2/|T|} \leq 1$.

In conclusion, (1) remains established in both cases. □

Remark: A priority queue for storing n messages can be laid out in area $O(n \log N)$ using a systolic implementation[Lei79]. If $d(M)$ is constant, then by Lemma 7 the priority queues could be stored in the leaves without changing the network's $O(N \log^2 N)$ area bound.

4 Deterministic On-line Routing on the Sorting Fat-tree

The partition of messages by peak level in the pruned butterfly may unnecessarily serialize the routing of subsets of messages which use different channels and hence could be routed simultaneously. The sorting fat-tree, to be described next, circumvents this problem by first bringing all messages to their peaks and storing them. Unlike the pruned butterfly, a given node v of the sorting fat-tree can then reorganize *all* of the messages with peak v for more efficient transmission down to their destinations. This strategy leads in certain cases to an optimal routing time of $O(\lambda \log N)$. However, the strategy requires that all messages be present in the routing network simultaneously and hence limits the class of message sets that can be handled.

In Subsection 4.1 we describe the main component of each node of the fat-tree: a new type of sorting circuit. Subsection 4.2 describes the sorting fat-tree the network itself, and Subsection 4.3, the routing algorithm.

4.1 Flexible Sorters

The main task of the sorting circuit placed at node v of the fat-tree is to reorganize the set of messages with peak v. Roughly speaking, for the entire algorithm to route in time proportional to the load factor, node v must be able to sort in time proportional to the size of its peak set. Qualitatively, we refer to a circuit that sorts in time proportional to its input size as a *flexible* sorter. Although VLSI sorting has been investigated extensively, the circuits proposed in the literature are for sequences of a fixed length l. Since they do not yield better performance on sequences of length $r < l$, they are not flexible. In this subsection, we design a flexible sorter with the properties summarized by the following theorem.

Theorem 2 *Let $\gamma \geq 1$ be a constant and let $l = O(s^\gamma)$. Then there is a VLSI circuit that, for any r such that $s \leq r \leq l$, can sort r words of b bits in time $O((b + \log s)r/s)$, and that can be laid out in a rectangle of height $O(s)$ and length $O(s\lceil b/\log s\rceil + (l/s)(b + \log l))$.*

We use an extension of Columnsort[Lei85a], reviewed in Lemma 8 below, to sort $r = ws$ words by a number of phases. Each phase consists of w sorting operations on sequences of size s, and of one permutation of ws words. To sort s words, we use the area-time optimal circuit of [BP85], reviewed in Lemma 9 below.

Lemma 8 *Let P be a two-dimensional array with s rows and r/s columns. Then P's elements can be sorted into column major order by an algorithm consisting of*

$$\left(\frac{\log r}{\log(s/2)}\right)^{2/\log(3/2)}$$

phases. In each phase, the columns of P are sorted and a permutation is applied to the entries of P. At a given phase, the permutation depends only upon r and s.

Proof: Recall that Columnsort sorts an $s\times(r/s)$ array of words by sorting the columns four times and, after each time, permuting the entries of the array according to a fixed pattern. A sufficient condition for Columnsort to work properly is that $s \geq 2^{1/3}r^{2/3}$. If this relation is not satisfied, one can resort to a recursive application of Columnsort. The input array P is partitioned into blocks of consecutive columns, with each block containing $2^{1/3}r^{2/3}$ elements. (Here and below, we ignore for simplicity that some quantities may not be integer; suitable adjustments could make the analysis rigorous without altering the essence of the result.) These blocks are then treated as "virtual columns" and are sorted by a recursive call to Columnsort, unless their size is s or less. A straightforward analysis shows that, at the k^{th} level of recursion (the main call being at level 0), the size of the sorting problems is $2^{(1-(2/3)^k)}r^{(2/3)^k}$. This size becomes $\leq s$ for $k = k^* = (\log\left(\frac{\log r}{\log(s/2)}\right))/\log(3/2)$. As the recursion tree has degree 4, there are $4^{k^*} = ((\log r)/\log(s/2))^{2/\log(3/2)}$ leaves. A leaf call corresponds to sorting each of the r/s columns of the $s \times (r/s)$ array. The number of permutation steps between the sorting phases corresponding to two consecutive leaf calls is equal to the height in the recursion tree of their lowest common ancestor. By combining these consecutive permutation steps into a single permutation step, we obtain an algorithm with the desired structure. $\square$

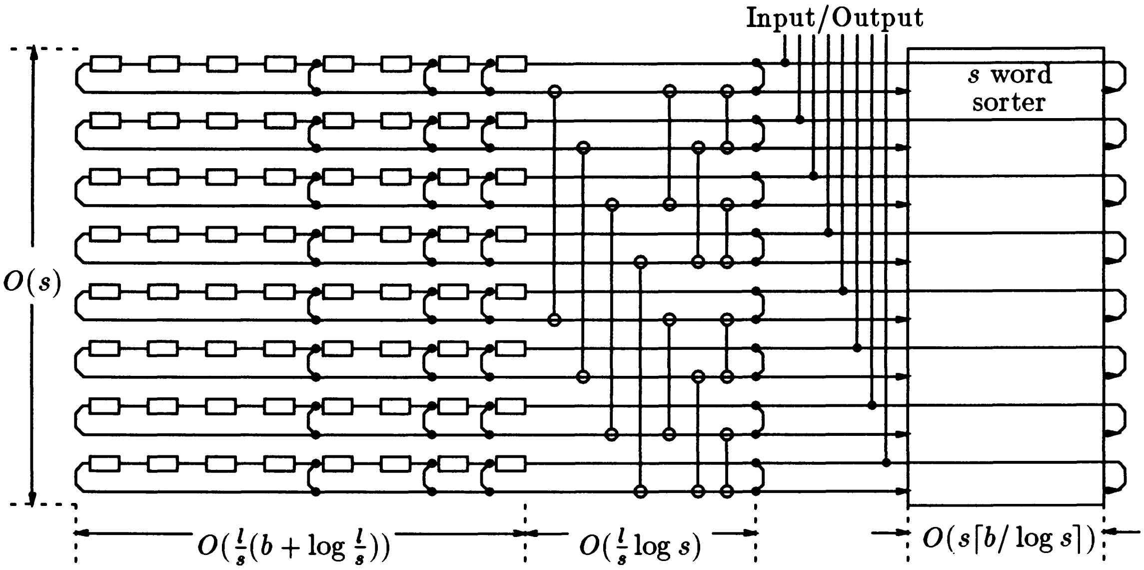

Figure 4: A flexible sorter with $s = 8$ and $l = 64$

Lemma 9 *There is a VLSI circuit for sorting s words of length b in time $O(b + \log s)$ and area $O(s^2\lceil b/\log s\rceil)$. Furthermore the circuit fits in a rectangle of height $O(s)$ and length $O(s\lceil b/\log s\rceil)$.*

Proof: The paper [BP85] shows how to construct such a circuit to fit in an $O(s)$ by $O(s)$ square when $b = O(\log s)$. The area bounds for arbitrary b cited above are obtained without modifying the layout except to extend the registers that compare words and to stretch wires that connect them to the rest of the layout. □

We are now ready for the main proof of this subsection.

Proof of Theorem 2: To simplify the presentation, we assume s and l are powers of two. The circuit consists of the sorter for s words of Lemma 9 and a permuter for up to l words. The permuter consists of s cycles of l/s cells, interconnected by an Omega network [Law75] as shown in Figure 4. Each cell, represented by a small rectangular box in the figure, consists of two b-bit bidirectional shift registers and some associated circuitry. In constant time, the two registers can be exchanged or one loaded with the contents of the other. Each cell is $O(1)$ high and $O(b)$ long and is connected by a constant number of wires to its neighbors on either side. The cells of each cycle are labeled from 1 to l/s starting from the right. As illustrated, the cycles have switches that permit them to be closed off at any length that is a power of two up to l/s. The switches of the Omega network are one bit wide, and are represented in the figure by vertical lines joining open circles. Some additional registers not shown in Figure 4 are required to control the permutation routing. Attached to each switch of the Omega network is a shift register with $O(l/s)$ bits laid out horizontally parallel to one of the two cycles joined by the switch. Attached to each cell labeled with an even number between 2^{m-1} and 2^m is a shift register with $O(\log^2 l/s - m^2)$ bits, laid out horizontally between it and the neighboring cell.

The height of the circuit layout is proportional to s, the number of cycles. The length of the layout of each cycle is

$$O(bl/s + \sum_{m=1}^{\log l/s} 2^m(\log^2 l/s - m^2)) = O(bl/s + l/s \log l/s).$$

Since each stage of the Omega network has length $O(l/s)$, the entire Omega network has length $O((l/s)\log s)$. Adding in the length of the sorter from Lemma 9, we get $O(s\lceil b/\log s\rceil + (l/s)(b + \log l))$ for the length of the flexible sorter, as claimed.

We now describe how the flexible sorter implements the algorithm of Lemma 8. The r input words, each of length b, enter the circuit bit-serially in groups of size s on the s wires shown at the top of Figure 4. The words are shifted into the rightmost $\lceil r/s \rceil$ columns of the cell array. The cycles are then closed off at the next larger power of two, say w. From this point on, the algorithm consists of a constant number of alternating sorting steps and permuting steps. In a sorting step, each of the w columns of s words is passed through the sorter by a counterclockwise bit-serial rotation through the cycles. In a permuting step, the circuit simulates the Benes permutation network [Ben65] using essentially the same technique as the cube-connected cycles architecture [PV81].

Recall that when an n-line Benes network routes a permutation, each of the $2n \log n$ switches requires a control bit to tell it whether or not to exchange the words appearing at its inputs. Consequently our permuter requires control bits when it simulates an sw-line Benes network. To route a permutation of sw words, each switch of the Omega network needs $2w$ control bits to determine which pairs of words to exchange between the cycles it joins. Each cell with an even label between 1 and w needs $2 \log w$ control bits to determine which words to exchange with its odd neighbor. For any fixed permutation, the control bits can be precomputed in polynomial time as in [Wak68].

The set of permutations our flexible sorter must route is fixed at the time of the sorter's construction: the circuit must be able to execute Columnsort on sw words for any value of w a power of two between 1 and l/s. Since every such w is $O(s^{\gamma})$, by Lemma 8 at most a constant number of permutations, say β, are needed for each valid w. The control bits a Benes network would use to route these $\beta \log l/s$ permutations are precomputed and stored in the shift registers at the switches of the Omega network and between odd and even numbered cells in the array. In each register, the bits are arranged in $\log l/s$ fields, one for each valid w. The fields are in order from smallest to largest w, and within each field the bits are in the order that the β permutations are used by Columnsort. In an Omega network control register, each field has $2\beta w$ bits, giving a total of $\sum_{k=0}^{\log(l/s)} 2\beta 2^k = O(l/s)$ bits. In a control register at a cell labeled with an even number between 2^{m-1} and 2^m, each field has $2\beta \log w$ bits, for a total of $\sum_{k=m}^{\log l/s} 2\beta k = O(\log^2 l/s - m^2)$ bits.

After w is known and just prior to the first permuting step, all control registers are simultaneously shifted so that the appropriate field is at the front of the shift register. Then during the permuting steps the control bits can be read and used without delaying the data movement.

Let us now analyze the time taken by the flexible sorter. Shifting the input into the cell array takes time $O(br/s)$. Shifting the permutation control registers so the appropriate field is ready to be read takes time $O(\log^2 w)$ for the registers in the cell array and time $O(w)$ for the registers in the Omega network. Since each column can be sorted in time $O(b + \log s)$, each sorting step of the algorithm takes $O(w(b + \log s))$. The cube-connected-cycles method of executing a permuting step takes $O(wb + \log s)$ time. Observing that $w = O(r/s)$ and sorting steps dominate the other costs leads to the bound claimed. □

Henceforth we will refer to the circuit of Theorem 2 as a (b, s, l)-flexible sorter. A special case of the (b, s, l)-flexible sorter works in optimal time for its area over the entire range of input sizes.

Taking into account the known bisection-based lower bound for sorting[Tho80, Lei85a, Bil84], we have:

Corollary 2 *For $b = \log s + \Theta(\log s)$ and $l = O(s^2/\log s)$, a (b, s, l)-flexible sorter has area $O(s^2)$ and sorts in time $O((n/s)\log s)$, which is AT^2-optimal.*

4.2 The Sorting Fat-Tree

The sorting fat-tree is an N-terminal routing network whose structure is a hybrid of a fat-tree and a mesh. Groups of $\log^2 N$ terminals are interconnected by $\log N \times \log N$ two-dimensional meshes. Each terminal in a mesh is allocated a square region with side length $O(\log N)$. Thus each mesh occupies a square region with side length $O(\log^2 N)$. The $N/\log^2 N$ meshes are placed at the leaves of a fat-tree laid out in the H-tree style. (For convenience, we assume N and $N/\log^2 N$ are powers of two.) The structure of the node at the root of a subfat-tree with n terminals is different depending on whether n is an even or odd power of two. If n is an odd power of two, the node contains a $(b, \sqrt{2n}, n)$-flexible sorter whose $\sqrt{2n}$ outputs enter a channel of capacity $\sqrt{2n}$ connecting the node to its parent. If n is an even power of two, the node contains $(b, 2\sqrt{n}, n)$-flexible sorter whose $2\sqrt{n}$ outputs are multiplexed into a channel of capacity $\sqrt{n}$ connecting the node to its parent. Figure 5 illustrates two adjacent levels of the tree, with the flexible sorters labeled by their parameters.

The sorting fat-tree also has some auxiliary circuitry similar to that in the pruned butterfly fat-tree. A N-leaf tree structure with its leaves at the terminals permits the the sorting fat-tree to execute prefix computations as in Lemma 2. A second tree structure with its leaves at the internal nodes of the fat-tree permits the network to synchronize, in $O(\log N)$ time, operations at all internal nodes. Associated with each terminal are a pair of b-bit shift registers that can be compared and exchanged.

To analyze the area occupied by the sorting fat-tree, let ξ be a constant large enough so that a

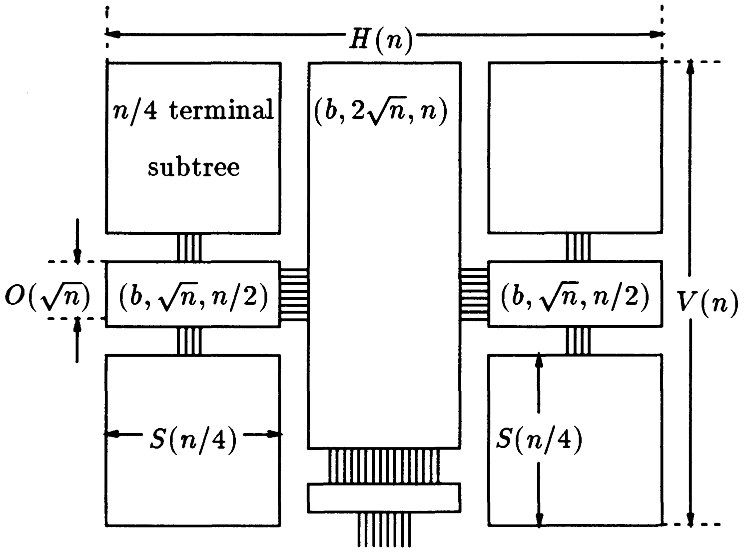

Figure 5: The layout of the sorting fat-tree

$(b, \sqrt{n}, n/2)$-flexible sorter and a $(b, 2\sqrt{n}, n)$-flexible sorter each fit in a rectangle of height $\xi\sqrt{n}$ and length $\xi\sqrt{n}\log N$, and mesh of $\log^2 N$ terminals fits in a square of side length $\xi \log^2 N$. Let $S(n)$ denote the length of one side of the (square) layout of a subtree of the fat-tree containing n terminals, where n is a power of four and $n \geq \log^2 N$. Assume inductively that $S(n/4) \leq \xi\sqrt{n/4}\log(Nn/4)$. Clearly $S(n)$ is the maximum of $V(n)$, the vertical length, and $H(n)$, the horizontal length of the layout. Referring to Figure 5, one can see that $V(n) \leq \max(\xi\sqrt{n}\log N, \xi\sqrt{n}+2S(n/4))$ and $H(n) \leq \xi\sqrt{n} + 2\max(S(n/4), \xi\sqrt{n}\log N)$. Applying the induction hypothesis and some straightforward calculations shows that $S(n) \leq \max(H(n), V(n)) \leq \xi\sqrt{n}\log(Nn)$, thus reestablishing the induction hypothesis. The entire network fits in a square region of side length $2\xi\sqrt{N}\log N$, so its area is $O(N\log^2 N)$.

4.3 Routing Algorithm

The algorithm of this section works for any message set M of constant degree. For simplicity, we describe the special case where M is a partial permutation; the extension to constant degree message sets is straightforward. The following list of steps outlines the routing algorithm.

1. Messages with source and destination in the same mesh are routed by a standard technique.
2. Messages that must be sent between meshes are partitioned into $\log N$ batches.
3. Within each mesh, messages are reorganized into row major order by batch number.
4. The batches are routed to their peaks consecutively.
5. All the messages with the same peak are sorted in order of destination.
6. The messages are again partitioned into $\log N$ batches.
7. The batches are routed down to their destination meshes.
8. Within each mesh, the messages are routed to their final positions.

Step 1 can be accomplished in $O(\log N)$ time by standard mesh routing techniques based on sorting and prefix computation. Step 2 is more involved. We will first describe the partition, then how the network computes it.

Let $\overline{M}$ be the set of messages that must be routed between meshes. We denote by $\ell_a(\overline{M}, c)$ the load placed by a message set $\overline{M}$ on a channel c during the source-to-peak movement of the message set. For a channel c that is k levels down from the root of the fat-tree, the ascending load factor $\lambda_a(\overline{M}, c)$ is $\ell_a(\overline{M}, c)/\sqrt{N/2^k}$. Let M_i be the set of messages in $\overline{M}$ with their peak at level i ($0 \leq i < \log N - 2\log\log N$). Assign to each message in M_i a *rank* between 0 and $|M_i| - 1$ according to increasing order of the message sources. The set M_i is then partitioned according to rank into $M_{i0} \cup M_{i1} \cup \cdots \cup M_{i(\log N-1)}$, where M_{ij} is the set of messages in M_i whose rank modulo $\log N$ is j. Then the $\log N$ batches of the partition are given by $M_{*j} = \cup_i M_{ij}$, for each $0 \leq j < \log N$. This partition is useful because it divides the work of routing $\overline{M}$ roughly equally among the batches, as shown by the following lemma.

Lemma 10 *For each* $0 \leq j < \log N$, $\lambda_a(M_{*j}) = O(\lceil \lambda_a(\overline{M}) / \log N \rceil)$.

Proof: Let us first show that, except for $\log N$ messages, the load placed by a batch on any given channel is a factor of $\log N$ smaller than the load placed by the full message set on the channel. The statement of the lemma will then follow trivially from the definition of load factor. Let c_k be a level k channel ($k \le \log N - 2\log\log N$, since there are no channels within the meshes). Only the messages with peak levels smaller than k can affect the load a batch places on c_k, so for every j,

$$\ell_a(M_{*j}, c_k) = \sum_{i=0}^{k-1} \ell_a(M_{ij}, c_k). \tag{4}$$

Since all messages in M_i that must cross c_k have consecutive ranks, our choice of M_{ij} as those with rank $j (\mathrm{mod}\ \log N)$ amounts to selecting every $\log N^{th}$ message that must cross c_k. Thus

$$\ell_a(M_{ij}, c_k) = \left\lceil \frac{\ell_a(M_i, c_k)}{\log N} \right\rceil < 1 + \frac{\ell_a(M_i, c_k)}{\log N}. \tag{5}$$

Combining (4) and (5), we have:

$$\ell_a(M_{*j}, c_k) < k + \frac{1}{\log N} \sum_{i=0}^{k-1} \ell_a(M_i, c_k) < \log N + \frac{\ell_a(\overline{M}, c_k)}{\log N}.$$

Since every channel has capacity at least $\log N$,

$$\lambda_a(M_{*j}, c_k) = \frac{\ell_a(M_{*j}, c_k)}{\sqrt{N/2^k}} < \frac{\log N}{\sqrt{N/2^k}} + \frac{\ell_a(\overline{M}, c_k)}{\log N \sqrt{N/2^k}} = O(\left\lceil \frac{\lambda_a(\overline{M}, c_k)}{\log N} \right\rceil).$$

□

To compute the partition of $\overline{M}$ of Lemma 10, the network first labels each message with its peak level i. This is a simple matter of comparing the source and destination bits and can be done for all messages in parallel in $O(\log N)$ time. The network then prepends i to the messages' destination fields to simplify later routing operations. Computing each message's rank within the appropriate M_i can be done with a prefix operation for each M_i. Since each prefix operation takes $O(\log N)$ time, all the ranks, and hence all the batch numbers can be assigned to the messages in $O(\log^2 N)$ time.

The purpose of Step 3 is to prepare the messages so that when they are routed upward, the entire channel will be busy at once: there will be no gaps between messages. Since each mesh holds

$\log^2 N$ terminals and the capacity of a channel is the square root of the number of terminals below it, the channel that joins a given mesh to the fat tree has one wire for each column of the mesh. Regardless of the mesh's orientation in the layout, for the purposes of defining row major order, we will consider top of the mesh to be the side to which the channel is attached. Steps 3 and 8 are similar to Step 1 and can be accomplished in $O(\log N)$ time.

During Step 4 the batches of messages are routed in order from their source meshes up the tree to their peaks. The nodes of the fat-tree operate in lockstep under the control of a global synchronizer. The step consists of a sequence of stages numbered consecutively. In the even stages, each node at an even level receives the messages of a given batch from its children, reorganizes them, and sends them to its parent. In the odd stages, nodes at odd levels do the same. A stage is completed only when all the nodes have finished their jobs.

Before the messages of a given batch are sent across any channel in the fat-tree, they are grouped for efficient transmission into sets of size $\text{cap}(c)$ called *waves*. Each wave is sent across the channel in time proportional to the message length, with one message on each wire of the channel. A set B of messages is sent across channel c in time $O((|B| \log N)/\text{cap}(c)) = O(\lambda(B, c) \log N)$.

At each stage, after the node has received all messages in the current batch from both children, it separates messages that have their peak at the node from those that must be sent up to the parent. This is easily accomplished by sorting on the peak field previously prepended to the message destinations. As the messages exit the sorter, they are already properly organized into waves. Those destined for the parent are sent out immediately; those currently at their peaks are shifted into the unused portion of the flexible sorter's permuter, where they remain until Step 5.

A fat-tree node with parent channel c can execute the sorting and transmission operations on batch j in time $O(\lambda_a(M_{*j}, c) \log N)$. Thus by Lemma 10 each stage takes $O(\lceil \lambda_a(\overline{M})/\log N \rceil \log N)$ time. The number of stages is proportional to the height of the fat-tree plus the number of batches, or $O(\log N)$. Hence Step 4 takes time $O(\lambda_a(\overline{M}) \log N + \log^2 N)$.

In Step 5 the flexible sorter in each node is applied to all the messages that have their peaks at that node, taking time $O(\lambda(\overline{M}) \log N)$. Partitioning $\overline{M}$ into $\log N$ batches for the downward movement is simpler than the partitioning in Step 2. No prefix computations are required since the messages are already separated by peak level, and assigning batch numbers is a simple matter of marking each message with its position in the sorted sequence modulo $\log N$. Sorting the messages again by batch number completes Step 6 in time $O(\lambda(\overline{M}) \log N)$. Step 7 is symmetric with Step 4 and takes time $O(\lambda_d(\overline{M}) \log N + \log^2 N)$, where λ_d is the load factor of the peak-to-destination movement of $\overline{M}$. Since λ_a and λ_d are bounded by λ, we have

Theorem 3 *The sorting fat-tree of N terminals can route on-line a set M of $O(\log N)$-bit messages with constant degree and load factor $\lambda(M)$ in time $O(\lambda(M) \log N + \log^2 N)$ and area $O(N \log^2 N)$.*

5 Lower Bounds

Let R be an N-terminal router of area A. By decomposition tree techniques[BL84, Lei85b], the terminals of R can be labeled in such a way that the time T to route M satisfies the tradeoff $AT^2 = \Omega(b^2\eta^2(M)N)$, where $\eta(M)$ is the reference load factor. This lower bound captures the bandwidth constraints imposed by a certain set of cuts of the network. It is natural to ask whether there is a router that can achieve this lower bound, delivering any message set with performance $AT^2 = O(b^2\eta^2(M)N)$. The answer is negative, at least for a wide class of routers, as stated by Theorem 5 below.

We say that a routing algorithm is *content preserving* if the body of each message is not modified while the message travels from source to destination. More formally, we require that each bit of the message body trace a source-to-destination path, with the property that if any two paths share an edge the corresponding bits traverse that edge at different times.

Most routing algorithms proposed in the literature, including those in the present paper, are content preserving. (An exception is [Rab89], but there packet splitting and recoding is for the sake of fault-tolerance, not efficiency.) Content-preserving routers satisfy the following lower bound:

Theorem 4 *Let R be an N-terminal routing network with a content-preserving routing algorithm (either off-line or on-line). Then for each value of b and each $\eta \leq \sqrt{N}$ there exists a set M of b-bit messages with reference load factor $\Theta(\eta)$, such that the time T to route M on R and the area A of R satisfy the tradeoff $AT^2 = \Omega(b^2\eta^2 N \log^2(N/\eta^2))$.*

To prove Theorem 4, we first develop a few lemmas, for which we need a number of definitions. Recall that a graph (V, E) is an (α, β)-expander if for any $S \subseteq V$ such that $|S| \leq \beta|V|$, $|\Gamma(S) - S| \geq \alpha|S|$, where $\Gamma(S)$ is the set of vertices adjacent to some vertex in S. Let $\mathrm{EXP}(n, \alpha, \beta)$ denote an n-vertex (α, β)-expander. We also recall [Har69, pp. 21–23] that given two graphs $G_1 = (U, E_1)$ and $G_2 = (V, E_2)$, their cross product, denoted $G_1 \times G_2$, is the graph $(U \times V, E)$ where $\{(u_1, v_1), (u_2, v_2)\} \in E$ whenever either $u_1 = u_2$ and $\{v_1, v_2\} \in E_2$ or $v_1 = v_2$ and $\{u_1, u_2\} \in E_1$.

Lemma 11 *If G_1 and G_2 are (α, β)-expanders, then $G_1 \times G_2$ is an $(\alpha/4, \beta^2/2)$-expander.*

Proof: First we develop some notation. Let $G_1 = (U, E_1)$ and $G_2 = (V, E_2)$, where $|U| = p$ and $|V| = q$. Let $G = G_1 \times G_2 = (W, E)$, where $W = U \times V = \{(u_i, v_j) : u_i \in U$ and $v_j \in V$, for $0 \leq i < p$ and $0 \leq j < q\}$. For $X \subseteq W$ and $0 \leq k < q$, let $\gamma_k(X) = \{(u_i, v_j) \in X : j = k\}$. If one views G as q copies of G_1, where homologous vertices of the copies are connected according to the pattern of G_2's edges, then $\gamma_k(W)$ is the vertex set of k^{th} copy. A restricted form of adjacency in G, namely adjacency through copies of G_1's edges, is defined as follows. For $X \subseteq W$, $\Gamma_1(X) = \{(u_i, v_j) \in W : (u_i, v_j)$ is adjacent to a vertex in $\gamma_j(X)\}$. Clearly $\Gamma_1(X) \subseteq \Gamma(X)$. Mentally reversing the roles of G_1 and G_2, let $\rho_h(X) = \{(u_i, v_j) \in X : i = h\}$, for each $0 \leq h < p$, and let $\Gamma_2(X) = \{(u_i, v_j) \in W : (u_i, v_j)$ is adjacent to a vertex in $\rho_i(X)\}$.

Let $S \subseteq W$ such that $|S| \leq \beta^2|W|/2$. We will isolate the portions of S that do not expand well under the adjacency functions Γ_1 and Γ_2. Let $I = \{i : |\rho_i(S)| > \beta q\}$ and let $J = \{j : |\gamma_j(S)| > \beta p\}$. The set I (respectively, J) is the indices of the copies of G_2 (G_1) that contain too large a part of S to be guaranteed expansion through G_2's (G_1's) edges. Let $S_{\mathrm{lost}} = \{(u_i, v_j) \in S : i \in I$ and $j \in J\}$.

S_{lost} is the part of S that doesn't expand either through G_1's edges or through G_2's. Clearly

$$|S_{\text{lost}}| \leq |I||J| \leq \frac{|S|}{\beta q}\frac{|S|}{\beta p} = \frac{|S|}{\beta^2|W|}|S|. \tag{6}$$

We now turn our attention to $S - S_{\text{lost}}$. Let $S_1 = \{(u_i, v_j) \in S : j \notin J\}$ and let $S_2 = \{(u_i, v_j) \in S : i \notin I\}$. Since for each $j \notin J$, $|\gamma_j(S_1)| \leq \beta p$, by the expansion property of G_1, $|\Gamma_1(S_1) - S_1| \geq \alpha|S_1|$. From the definitions of Γ_1 and S_1, one can see that $\Gamma(S) - S \supseteq \Gamma_1(S_1) - S_1$. Similarly, $|\Gamma_2(S_2) - S_2| \geq \alpha|S_2|$ and $\Gamma(S) - S \supseteq \Gamma_2(S_2) - S_2$. It follows that

$$\begin{aligned}
|\Gamma(S) - S| &\geq \max(|\Gamma_1(S_1) - S_1|, |\Gamma_2(S_2) - S_2|) \\
&\geq \alpha \max(|S_1|, |S_2|) \\
&\geq \frac{\alpha}{2}|S_1 \cup S_2| = \frac{\alpha}{2}|S - S_{\text{lost}}|.
\end{aligned}$$

From (6) and the upper bound on $|S|$ we can therefore conclude

$$|\Gamma(S) - S| \geq \frac{\alpha}{2}(|S| - \frac{|S|}{\beta^2|W|}|S|) = \frac{\alpha}{2}(1 - \frac{|S|}{\beta^2|W|})|S| \geq \frac{\alpha}{4}|S|. \qquad \square$$

Given a graph G whose vertices are identified with the terminals of a routing network R, we denote by $M(G)$ the set containing for each edge $\{u, v\}$ of G a message from u to v. Also, let area(G) denote the (minimum) layout area of graph G. We are now ready to state a result that links the layout area of graphs to the AT^2 measure of content-preserving routers.

Lemma 12 *Let G be a graph of constant degree. If a content-preserving router of area A can route $M(G)$ in time T, then*

$$AT^2 = \Omega(\textit{area}(G \times \textit{EXP}(b, \alpha, \beta))),$$

where b is number of bits in the body of a message in $M(G)$.

Sketch of Proof: Because R's routing algorithm is content-preserving, the paths traversed by the bits of the messages of $M(G)$ form an embedding of b copies of G in R, with congestion at most T. By a variant of a technique due to Thompson [Tho80], such an embedding can be converted to a layout of area $O(AT^2)$ for $G \times \text{EXP}(b, \alpha, \beta)$, whence the statement of the lemma.

We now outline the construction of the layout of $G \times \mathrm{EXP}(b, \alpha, \beta)$. Assume that R is laid out in two layers on a square grid $\mathcal{G} = \{(i,j) : 0 \leq i < I, 0 \leq j < J\}$, with $IJ = A$. The layout of $G \times \mathrm{EXP}(b, \alpha, \beta)$ will have six layers on a square grid $\widehat{\mathcal{G}} = \{(h, k : 0 \leq h < IT, 0 \leq k < JT\}$. Intuitively, $\widehat{\mathcal{G}}$ is viewed as the interleaving of T stretched copies of $\mathcal{G}$. Laid out on the t^{th} copy of $\mathcal{G}$ are the wires of R that are active at time t in delivering some bit of $M(G)$. More precisely, if a horizontal wire of R is used at time t (for $1 \leq t \leq T$) to move some bit from point (i, j) to $(i, j+l)$ on grid $\mathcal{G}$, then a wire is placed in $\widehat{\mathcal{G}}$ from $(iT+t, jT+t)$ to $(iT+t, (j+l)T+t)$. A similar mapping is defined for vertical wires.

In the resulting layout, for every sequence of wires of R traversed by a bit going source to destination, there is a corresponding (disconnected) sequence of stretched wires. The stretched wire sequences are converted to paths by adding an extra edge (on a different layer) between each pair of adjacent stretched wires. Thus, we have obtained a three-layer layout on $\widehat{\mathcal{G}}$ of the edges of b copies of G. For each copy, we then connect together the endpoints of the edges that are incident upon the same vertex of G, and so obtain a four-layer layout of b copies of G. Next we construct expanders on each group of b homologous copies of a vertex of G. As these copies are placed in a $T \times T$ region of $\widehat{\mathcal{G}}$, and $b \leq T$, the expander connections can be added on two extra layers without increasing the layout area. □

The graph that will play the role of G in Lemma 12 is the cross product of an expander and $\mathrm{MOT}(n)$, the n-vertex mesh-of-trees [Lei81], to be defined next. Let $\eta \leq \sqrt{N}$ be a power of two and let n be the largest integer of the form $3 \cdot 2^{2k} - 2^{k+1}$ not larger than N/η^2. Place 2^{2k} of the n vertices in the even rows and columns of a $2^{k+1} \times 2^{k+1}$ grid. These vertices are called leaves. In each occupied row and column, construct a complete binary tree on the 2^k leaves by adding vertices and edges as needed. The 2^{k-1} internal nodes of a tree are placed at the odd grid positions. Thus an examination of the grid points in an even row from left to right or in an even column from top to bottom, yields an in-order traversal of the corresponding tree. The vertices of $\mathrm{MOT}(n)$ are named by their association with the grid points, which are themselves labeled in "shuffle-major" order: the grid point in row r and column c, for $0 \le r, c < 2^{k+1}$, is labeled with the integer whose binary representation is the bits of r interleaved with the bits of c.

Lemma 13 *Under an appropriate naming of the vertices, the reference load factor of $MOT(n) \times EXP(\eta^2, \alpha, \beta)$ is $\Theta(\eta)$.*

Proof: First we show that the reference load factor of $\mathrm{MOT}(n)$ is constant. Recall the binary tree used to define the reference load factor. The labels on the leaves of any complete subtree of height h correspond to the labels in a rectangular region of the grid containing the mesh of trees. The number of edges of $\mathrm{MOT}(n)$ crossing the boundary of any such rectangle is proportional to the length of the perimeter. But the perimeter has length $\Theta(2^{h/2})$, which is proportional to the weight of the edge leaving the root of the subtree. Thus the reference load factor is constant.

When assigning names to vertices of $\mathrm{MOT}(n) \times \mathrm{EXP}(\eta^2, \alpha, \beta)$, place the expander part of each vertex's name in the least significant position. Thus, each η^2-vertex expander gets mapped to an η^2-leaf subtree when computing the reference load factor. The statement of the lemma follows. □

Lemma 14 *If H is any m-vertex expander, then area$(MOT(n) \times H) = \Omega(m^2 n \log^2 n)$.*

Proof: The graph $\mathrm{MOT}(n) \times H$ is a supergraph of the expander connected mesh-of-trees $Q_{m,2^k}$ defined in [BL84]. The lower bound on wire area derived there applies directly. □

We are now ready to combine the lemmas into a proof of Theorem 4. The message set M in question is the one corresponding to the graph $\mathrm{MOT}(n) \times \mathrm{EXP}(\eta^2, \alpha, \beta)$. By Lemma 13, the reference load factor of M is $\Theta(\eta)$. By Lemma 12, any area A router that routes M in time T satisfies

$$AT^2 = \Omega(\mathrm{area}(\mathrm{MOT}(n) \times \mathrm{EXP}(\eta^2, \alpha, \beta) \times \mathrm{EXP}(b, \alpha, \beta))).$$

By Lemma 11,

$$AT^2 = \Omega(\text{area}(\text{MOT}(n) \times \text{EXP}(\eta^2 b, \alpha/4, \beta^2/2))).$$

Finally, by Lemma 14, and the fact that $n = \Theta(N/\eta^2)$, $AT^2 = \Omega(\eta^4 b^2 n \log^2 n) = \Omega(\eta^2 b^2 N \log^2(N/\eta^2))$.
□

6 Concluding Remarks

Our results on routing can be interpreted by relating them to known bounds on the layout area of graphs. It is well known[BL84] that a graph with a (minimum $\sqrt{2}$-) bifurcator of size F has layout area at least $\Omega(F^2)$ and at most $O(F^2 \log^2(N/F))$. Moreover, each of these bounds is achieved by some graph. Therefore, the bifurcator determines the area only to within a factor of $\log^2(N/F)$.

An analogous situation holds for the reference load factor as a predictor of the AT^2 measure of routing complexity, albeit with a few more conditions. Suppose we restrict our attention to content-preserving routers and constant-degree message sets with $\Theta(\log N)$-bit messages and reference load factor η, where $\Omega(\log N) \leq \eta \leq O(N^{1/2-\epsilon})$, for some fixed constant $\epsilon > 0$. Then the AT^2 measure for routing a message set is at least $\Omega(\eta^2 N \log^2 N)$, by arguments from [Lei85b], and at most $O(\eta^2 N \log^2 N \log^2(N/\eta^2))$, by Theorem 3 on the sorting fat-tree. It is easy to construct message sets for which $AT^2 = \Theta(\eta^2 N \log^2 N)$, but there are also some for which $AT^2 = \Theta(\eta^2 N \log^2 N \log^2(N/\eta^2))$, as established in Theorem 4. Therefore the reference load factor determines the AT^2 measure only to within a factor of $\log^2(N/\eta^2)$.

It is natural to ask whether some parameter in addition to the load factor could lead to tight characterization of the area-time complexity of routing. However, even within the load factor framework, the conditions on the above analogy suggest several avenues for further research. With regard to the lower bound, it would be interesting to see if the restriction to content-preserving routers could be removed. However, the main problem left open is to design an N-terminal network that can route any message set in $O(\eta \log N)$ time and $O(N \log^2 N)$ area, matching the lower bound of Theorem 4. We conjecture that such an optimal router would combine features from both of our fat-trees. To route message sets with constant load factor in logarithmic time, a network should, like the pruned butterfly, have tree nodes with only constant depth. Moreover, the routing algorithm should simultaneously deal with messages that have peaks at different levels, as in our sorting fat-tree.

References

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15^{th} Annual Symposium on the Theory of Computing,* Boston, Massachusetts, pages 1–9, April 1983.

[Ale82] R. Aleliunas. Randomized parallel communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 60–72, August 1982.

[Ben65] V. E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, New York, 1965.

[Bil84] G. Bilardi. *The Area-time Complexity of Sorting.* PhD thesis, University of Illinois, Urbana-Champaign, Illinois, December 1984. Coordinated Science Laboratory Report R-1024.

[BL84] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[BP85] G. Bilardi and F. P. Preparata. A minimum area VLSI network for $O(\log N)$ time sorting. *IEEE Transactions on Computers*, C-34(4):336–343, April 1985.

[BP89] G. Bilardi and F. P. Preparata. Size-time complexity of boolean networks for prefix computations. *Journal of the ACM*, 36(2):362–382, April 1989.

[DS82] E. Dekel and S. Sahni. Binary trees and parallel scheduling algorithms. *IEEE Transactions on Computers*, C-32(3):307–315, March 1982.

[GL85] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In *Proceedings of the 26^{th} Annual Symposium on the Foundations of Computer Science*, pages 241–249, October 1985.

[GL89] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Randomness and Computation*, pages 345–374. JAI Press, Inc., 1989.

[Gre90] R. I. Greenberg. The fat-pyramid: A robust network for parallel computation. In W. J. Dally, editor, *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 195–213, 1990.

[Har69] F. Harary. *Graph Theory.* Addison-Wesley, Reading, MA, 1969.

[HB88] K. T. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded-degree networks. In *Proceedings of the* 26^{th} *Annual Allerton Conference on Communication, Control and Computation, Monticello, Illinois*, September 1988.

[Her89] K. T. Herley. Efficient simulations of small shared memories on bounded degree networks. In *Proceedings of the* 30^{th} *Annual Symposium on the Foundations of Computer Science,* Research Triangle, North Carolina, pages 390–395, October 1989.

[Her91] K. T. Herley. A note on the token distribution problem. *Information Processing Letters*, 38(6):329–334, June 1991.

[Law75] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, C-24(12):1145–1155, December 1975.

[Lei79] C. E. Leiserson. Systolic priority queues. Technical Report CMU-CS-79-115, Department of Computer Science, Carnegie-Mellon University, April 1979.

[Lei81] F. T. Leighton. *Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques for VLSI.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1981.

[Lei84] F. T. Leighton. New lower bound techniques for VLSI. *Mathematical Systems Theory*, 17:47–70, 1984.

[Lei85a] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, April 1985.

[Lei85b] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–900, October 1985.

[LM89] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults on multibutterflies. In *Proceedings of the* 30^{th} *Annual Symposium on the Foundations of Computer Science,* Research Triangle, North Carolina, pages 384–389, October 1989.

[LMR88] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the* 29^{th} *Annual Symposium on the Foundations of Computer Science,* White Plains, New York, October 1988.

[NMB83] D. D. Nath, S. N. Maheshwari, and P. C. P. Bhatt. Efficient VLSI networks for parallel processing based on orthogonal trees. *IEEE Transactions on Computers*, C-32(6):569–581, June 1983.

[Pip84] N. Pippenger. Parallel communication with limited buffers. In *Proceedings of the* 25^{th} *Symposium on the Foundations of Computer Science*, pages 127–136, October 1984.

[PU87] D. Peleg and E. Upfal. The generalized packet routing problem. *Theoretical Computer Science*, 53:281–293, 1987.

[PU89] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18(2):229–243, April 1989.

[PV81] F. P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, May 1981.

[Rab89] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, April 1989.

[Ran87] A. G. Ranade. How to emulate shared memory. In *Proceedings of the* 28^{th} *Annual Symposium on the Foundations of Computer Science,* Los Angeles, California, pages 185–192, October 1987.

[Tho80] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Carnegie-Mellon University, August 1980.

[Upf84] E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31(3):507–517, July 1984.

[Upf89] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *Proceedings of the* 21^{st} *Annual Symposium on the Theory of Computing,* Seattle, Washington, pages 241–250, May 1989.

[Val82] L. G. Valiant. Scheme for fast parallel communication. *SIAM Journal on Computing,* 11(2):350–361, May 1982.

[VB81] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the* 13^{th} *Annual Symposium on the Theory of Computing*, pages 263–277, May 1981.

[Wak68] A. Waksman. A permutation network. *Journal of the ACM*, 15:159–163, 1968.