Advanced Research in VLSI 1991: UC Santa Cruz

Practical Schemes for Fat-Tree Network Construction

André DeHon
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
andre@ai.mit.edu

Abstract

As multiprocessor computer networks are scaled to support thousands and millions of processors, we must exploit locality in order to avoid uniform degradation in network performance. Fat-tree networks offer a topology that theoretically scales arbitrarily while allowing the exploitation of considerable locality. In this paper, I present a scheme for constructing practical fat-tree networks. Integrating expanders for redundant multipath switching networks, I incorporate fault-tolerance into the fat-tree network. I present primitive building blocks for the construction of these networks and describe how these building blocks can be synthesized using current technology. I also present organizational structures for composing these primitives into arbitrarily large networks. This synthesis results in a practical scheme for building large-scale, high-performance multiprocessor computer networks. With suitable locality and technology, a 786,432 processor network can route a message on the first attempt with over 70% probability when the network is fully loaded. The latency through the network from one endpoint to another is at most 320 ns. For more local connections, the network latency can be as small as 40 ns.

1 Introduction

The initial target for the Transit network is an indirect, pipelined, circuit-switched network arranged in a multistage shuffle-exchange, bidelta configuration. The Transit bidelta network utilizes an advanced three-dimensional stack packaging scheme, low-voltage swing signals, matched impedance drivers, and a novel routing component (RN1) to achieve high-performance, low-latency interconnection between network endpoints [5]. For moderately sized networks (*e.g.* 256 processors), this network structure provides uniform low-latency interconnection between all processors.

However, even with three-dimensional packaging technology, the distance between routing components grows as $\Theta(\sqrt{N})$ and the number of routing stages grows as $\Theta(\log N)$ as N, the number of processors in the system, is increased. Due to limitations in signal propagation speed, the increase in distance between routing components translates directly into longer network clock cycles. The increase in the number of routing stages effectively increases the number of pipeline stages between network nodes and hence increases the communication latency. In a bidelta configuration, all network interconnections thus degrade uniformly in performance as the network is scaled to support more and more processors.

This scaling problem, is not unique to the class of networks just described. This problem is inherent in all networks due to fundamental physical limitations. Signals cannot propagate faster than the speed of light. Components must be physically arranged in three-dimensional space, and wires of finite size must be routed in three-dimensional space. We cannot make components arbitrarily small. As the number of components in the system grows, we cannot keep them arbitrarily close to each other indefinitely.

Rather than accepting this uniform performance degradation as we scale upward to support more processors, we can optimize the interconnection scheme for locality. The idea is to take advantage of locality to minimize the impact of scaling on both aggregate network bandwidth and average network latency. In flat networks like the bidelta network, all processors are equally distant from each other. No locality can be exploited in the interprocessor communication. By allowing some nodes to be further away from each other in the network, other nodes can be placed closer to each other. With appropriate exploitation of the resulting locality, the average bandwidth is higher, and the average latency is lower than in a flat network configuration.

1.1 Fat-Tree

The network is based on Leiserson's fat-trees [9] [4] in order to achieve a reasonable amount of locality and adequate bandwidth. Fat-trees are organized with hierarchical, tree-structured interconnections. As with any tree, nodes with lower common ancestors are closer to each other. In a fat-tree the channel capacity increases regularly between internal tree routing nodes toward the root of the tree. This increase in capacity accommodates the greater amount of traffic which may need to travel through internal nodes closer to the root of the tree. Leiserson's theoretical results show that with the proper allocation of channel capacity at each tree level, fat-trees are volume-universal networks. That is, for a given volume of hardware, a fat-tree is nearly the best routing network one can build, and the fat-tree can simulate any other network with at most a polylogarithmic degradation in time [4].

1.2 Fault-Tolerance

We provide fault-tolerance in the fat-tree organization utilizing Leighton's randomly wired redundant paths and random path selection among equivalent paths. The multiple redundant paths through the network make it possible to avoid faulty network components. Randomly wiring the redundant paths increases the tolerance of multiple faults [7]. The random selection of a path among equivalent alternatives provides a computationally trivial, yet effective, mechanism for avoiding faulty portions of the network. [3] describes this scheme for achieving fault-tolerance in multistage routing networks in some detail.

1.3 Practical Construction

The fat-tree network described here is intended to be a practical network that can be physically realized. Leiserson's theoretical fat-tree results indicate that the architecture is physically realizable within the limitations of three-dimensional spatial construction [9]. The construction technology and organization provides a high degree of bandwidth [5] [3]. Leiserson's theoretical work also shows that this architecture is at least as scalable as any other network [9]. The fat-tree can be built from simple building blocks (see Sections 2.1 and 3) in highly replicatable structures (see Section 4), making the complexity of constructing large networks practical. Finally, the fault-tolerance inherent in the network construction makes the network reliable even when scaled [3] [8].

1.4 Network Constructed

The fat-tree network described here has redundant paths for increased fault-tolerance and routing performance [3] [8]. When the network is constructed using an RN1 style routing component (Section 2.1), the routing structure is an indirect, pipelined, circuit-switched network. The network is packaged using Transit three-dimensional stack technology (Section 2.2). Sections 3 and 5 describe the stack-based primitives used to achieve the basic fat-tree organization. Section 4 describes the basic geometry for arranging these stacks into fat-trees. Section 6 analyzes the performance of these networks.

2 Background

2.1 RN1

The RN1 routing component is used here as the lowest-level building block for fat-trees. Some of the details and specifics contained in this paper are closely related to RN1. However, the general construction topology and decomposition could be used with different routing components.

RN1 is a custom CMOS routing component currently under construction. It provides simple high-speed switching for fault-tolerant networks. RN1 has eight nine-bit wide input channels and eight nine-bit wide output channels. These channels provide byte-wide data transfer with the ninth bit serving as a signal for the beginning and end of transmissions. RN1 can be configured in either of two ways, as shown in Figure 1. The primary configuration is a 4×4 crossbar router with a dilation of two. In this configuration, all eight input channels are logically equivalent. Alternately, the component can be configured as a pair of 4×4 crossbars, each with four logically equivalent inputs and a dilation of one.

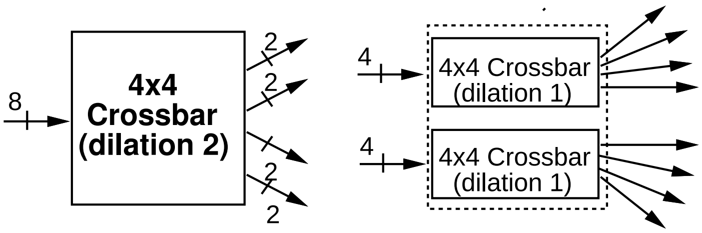


Figure 1: **RN1** Logical Configurations

Simple routing is performed by using the two bits of the first byte of a transmission to indicate the desired output destination. If an output in the desired direction is available, the data transmission is routed to one such output. Otherwise, the entire transmission is discarded. In either case, when the transmission completes, the RN1 routing component informs the sender of the connection status so that the sender will know whether or not it is necessary to retry the transmission. When both outputs in the desired output direction are available, the component randomly chooses which port to use.

To allow rapid responses to network requests, the RN1 routing component allows connections opened over the network to be reversed; that is, the direction of the connection can be reversed, allowing data to flow back from the destination to the source processor. The ability to reverse a network connection allows a processor requesting data to get its response quickly without requiring the processor it is communicating with to open a separate connection through the network.

RN1 is described further in [5], [3], and [10].

2.2 Packaging Technology

The basic packaging technology developed as a part of the MIT Transit project is the three-dimensional *stack*. A stack is composed of alternating layers, or planes, of components and printed circuit boards. In a routing stack, the component layers perform switching while the printed circuit boards interconnect routing components. The interconnection provided by the printed circuit boards controls the desired network topology. Figure 2 depicts roughly how multistage routing networks are mapped onto a routing stack. Figure 3 shows a more detailed cross-sectional view of the layers in a routing stack. [5] describes this stack structure in more detail.

The stack structure does have its physical limits. Current printed circuit board manufacturing technology limits the size of a side of a stack to under two feet. However, the stack structure gives us a dense and efficient way of interconnecting routing components to build higher-level building blocks for network construction.

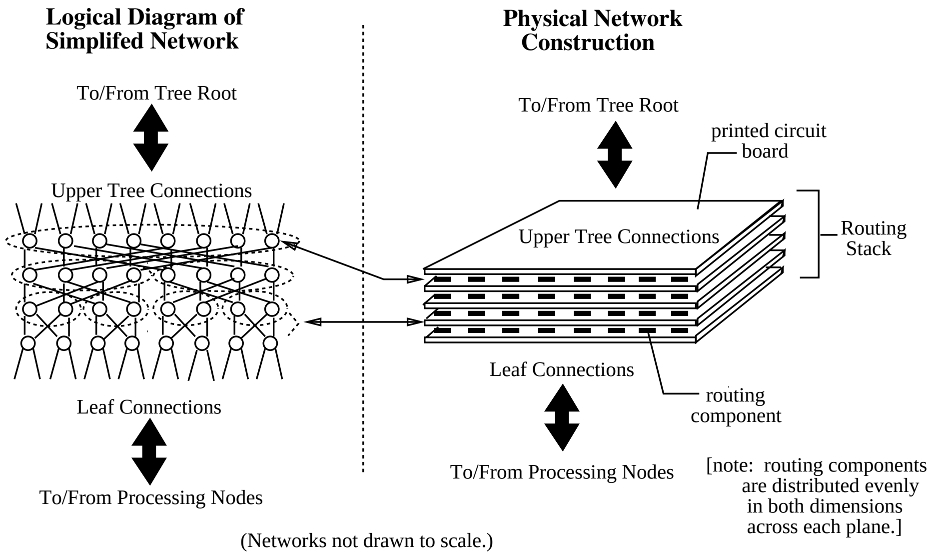


Figure 2: Network ⇔ Stack Mapping

3 Unit Tree

3.1 Organization and Composition

Using a routing component like RN1 with the stack packaging structure just described, we can build a few basic stack structures from which arbitrarily large networks can be built. The basic fat-tree stack primitive is known as the *unit tree*. Logically, a unit tree is organized as a piece of a fat-tree. Physically, each unit tree is composed of:

- a tree which routes upward or towards the root
- a tree which routes downward or toward the leaf nodes where the processors are located
- lateral crossovers between the up tree and down tree at each intermediate tree stage

Short-cut paths are provided in the upward routing tree so connections which must be made through intermediate nodes high in the tree can bypass some tree stages. When a connection is routed through the root of the fat-tree, the connection only traverses $\frac{4}{3}\log_4(N)$ routing stages between the two endpoints as compared to the $2\log_4(N)$ routing stages required without the short-cut paths.

The down routing tree implements a simple quaternary tree where each parent node forms the root of four subtrees (see Figure 4). Routing components in the down routing tree can simply route in one of four directions to select among the four subtrees rooted at the routing components.

The up routing tree routes connections to the appropriate height in the fat-tree. Routers in the up routing tree also direct connections in any of four directions. Three of these directions route into the down routing tree at each of the next

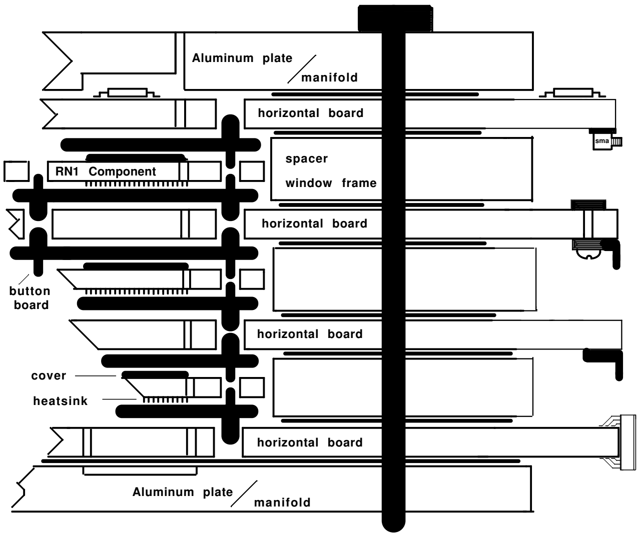


Figure 3: Cross-Section of Transit Routing Stack (Diagram courtesy of Fred Drenckhahn)

three tree stages. These connections from the up routing tree to the down routing tree make up the lateral crossovers. The fourth direction routes the connection further upward in the tree. Figure 5 shows the logical routing performed by an up routing stage, including the lateral crossovers.

Each unit tree implements three stages, or levels, of the fat-tree. The unit tree has four layers of routing components. The top three layers implement three stages of the down routing tree. The lowest layer implements a single stage of up routing. One stage of up routing is thus associated with each three down routing layers of fat-tree. Figure 6 shows the logical connections within a unit tree by showing connections among four routing components, each from a different stack layer.

Each unit tree has a single logical up routing channel leaving it for higher stages in the fat-tree and a single logical down routing channel entering from higher stages in the fat-tree. These connections to higher stages in the fat-tree enter and exit the top of the unit tree stack. Each unit tree also has $4^3 = 64$ logical up routing channels entering it from lower tree stages or endpoints and 64 logical down routing channels leaving it for lower tree stages. Connections to lower tree levels are made through the bottom of the unit tree stack.

3.2 Two Unit Tree Stacks

There are two unit tree configurations which match the organization just described and fit neatly into the physical constraints of stack packaging (Section 2.2).[1] Both of these unit trees connect logical channels from 64 subtrees in the fat-tree

[1] These "magic" sizes are actually the result of numerous constraints; for a complete description of the constraints which lead to these configurations, see [2].

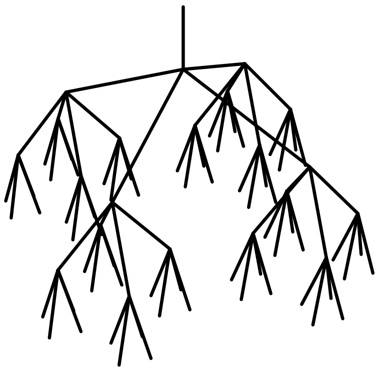
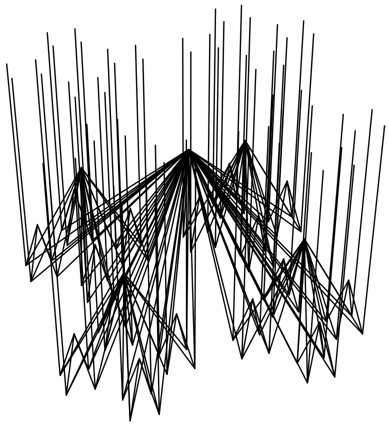



Figure 4: Connections in Down Routing Stages (left)

Figure 5: Up Routing Stage Connections with Lateral Crossovers (right)

Stack Layer	**Routing Components** $UT_{64\times 8}$	$UT_{64\times 2}$
0	64	16
1	64	16
2	48	12
3	32	8

Table 1: Unit Tree Component Summary

network to a single parent node (as shown logically in Figures 4 and 5). These unit trees differ in the number of physical network connections composing each logical channel. The $UT_{64\times 8}$ unit tree provides eight input and eight output network connections for each of its 64 subtrees, while the $UT_{64\times 2}$ provides two input and two output network connections for each of its subtrees. The $UT_{64\times 8}$ unit tree is roughly two feet square and two inches thick, while the $UT_{64\times 2}$ unit tree is roughly one foot square and two inches thick. Table 1 summarizes the number of components composing each of these unit trees. The number of routing components provides a metric for the cost and complexity of each of these stack building blocks. Table 2 characterizes these unit trees by summarizing their channel organization and capacity.

Processing nodes generally each have two inputs and two outputs to the network [3] to achieve fault-tolerance. This means that $UT_{64\times 2}$ unit trees can connect directly to processing nodes whereas $UT_{64\times 8}$ unit trees cannot. With the higher channel capacities, the $UT_{64\times 8}$ unit tree is preferable for building all of the portions of the fat-tree network which do not connect to the processors. Both of these unit trees are arranged with redundant paths for fault-tolerance as described in [3]. The final stage of down routing in the $UT_{64\times 2}$ unit tree is constructed out of RN1 routers configured in their dual 4×4 crossbar mode such

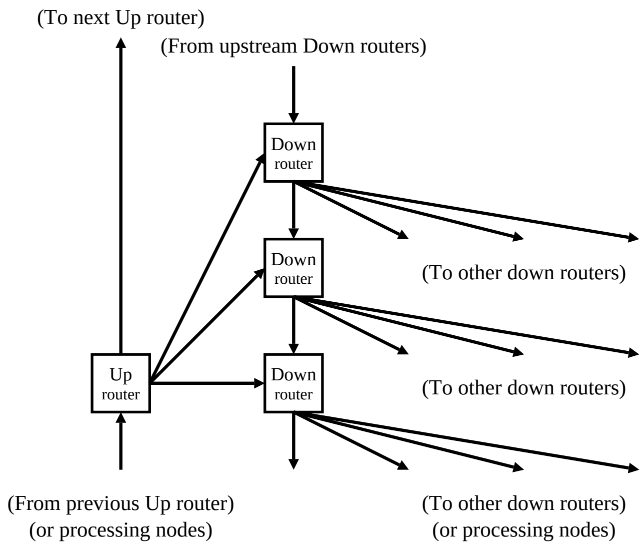


Figure 6: Cross-Section View of Unit Tree Layers

	Logical Channels	**Capacity**		**Total Channels**	
		$UT_{64\times 2}$	$UT_{64\times 8}$	$UT_{64\times 8}$	$UT_{64\times 2}$
Up from Leaves	64	8	2	512	128
Down toward Leaves	64	8	2	512	128
Up toward Root	1	128	32	128	32
Down from Root	1	128	32	128	32

Table 2: Unit Tree Channel Composition

that each of the outputs to a node comes from a different routing chip.

3.3 Volume-Universality

As mentioned in Section 1.1, one desirable property of fat-trees is their volume-universality. This property guarantees that a fat-tree can efficiently simulate any other network of comparable volume within a polylogarithmic time factor [9]. Additionally, volume-universality guarantees that larger and larger fat-trees can be created by simply adding tree levels and scaling the basic structure within the three-dimensional world.

To retain volume-universality in a quaternary fat-tree, the channel capacity must on average grow by a factor of $\sqrt[3]{16}$ on each successive level toward the tree root.[2] This growth rate follows directly from the fact that the volume of processors being interconnected increases by a factor of four at each successive tree level and the fact that the channel capacity available for routing communications increases proportionally to the surface area of that volume. Looking at the channel capacity summary in Table 2, we see that the capacity of a logical channel increases by a factor of 16 through the three tree levels contained in the unit tree. On average, this gives the desired channel capacity growth rate of $\sqrt[3]{16}$.

4 Hollow Cube

4.1 Fat-Tree Composition

Using $UT_{64\times 8}$ unit trees, we can easily construct large fat-tree networks. These fat-tree networks are built in stages of unit trees where each unit tree stage is composed of many unit tree stacks and makes up three tree levels of the fat-tree. Each additional stage of unit trees multiplies the number of processors supported by the fat-tree by 64.

At almost all points in the fat-tree, the channel capacity of a channel in a logical subtree is greater than the eight physical channels provided by the $UT_{64\times 8}$ unit trees. The larger channel capacity is handled by using multiple $UT_{64\times 8}$ unit trees to construct the logical parent nodes for such subtrees. The connections from the subtree unit tree stacks into the parent node unit tree stacks are dispersed as widely as possible; that is, physical channels from a single unit tree in the bottom tree are spread out as evenly as possible to all unit trees comprising the parent nodes. Spreading these redundant paths to and from subtrees as widely as possible among the parent trees further increases fault-tolerance. Proper dispersion makes it possible to remove entire unit tree stacks anywhere, except at the leaves, and still have a functionally complete fat-tree; performance is, of course, degraded when unit tree stacks inside the fat-tree network fail or are removed.

4.2 Examples

It is illustrative to consider some concrete examples of fat-tree composition.

4K A small fat-tree with 4096 nodes is constructed with two unit tree stages. $UT_{64\times 2}$ unit trees connect to the processing nodes. Each $UT_{64\times 2}$ unit tree connects 64 nodes into the network. To support the $64^2 = 4096$ nodes, 64 $UT_{64\times 2}$ unit trees are needed to connect to all the nodes. These 64 $UT_{64\times 2}$ unit trees are interconnected with 4 $UT_{64\times 8}$ unit trees. The 4 $UT_{64\times 8}$ unit trees form the logical root of this fat-tree.

[2] [9] and [4] do not necessarily prove universality of fat-trees with the generality used here. Given the structure of their proofs, it is likely that their work can be generalized in this manner.

256K With an additional stage of unit trees, the structure supports 64 times as many nodes (262,144). The basic structure of the previous example is replicated 64 times to support the additional processors. The additional $UT_{64\times 8}$ unit tree stage is added on top of the $4 \times 64 = 256$ $UT_{64\times 8}$ unit trees which form the middle unit tree stage. This root unit tree stage is composed of 64 $UT_{64\times 8}$ unit trees. The entire tree is thus built from $64^2 = 4096$ $UT_{64\times 2}$ unit trees and $256 + 64 = 320$ $UT_{64\times 8}$ unit trees.

4.3 Growth Rate

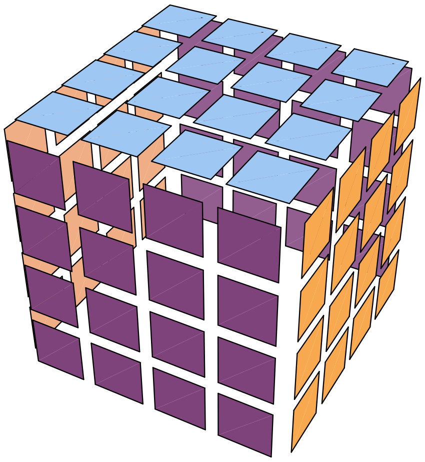
From channel capacity arguments, we see that the number of unit trees in a parent stage is always a factor of four smaller than the number of unit trees in the immediate child stage, as long as unit trees of the same size are used in both stages. At each internal unit tree stage, n unit trees form the root of each subtree. Sixty-four subtrees connect to one logical parent node composed of $\frac{64n}{4} = 16n$ unit trees. The area which must be interconnected thus increases by a factor of 16 at each stage.

Given these growth characteristics, there is no recursive, three-dimensional structure that keeps interstage interconnection distances constant. This is easily seen by noting that the number of components grows exponentially with the number of tree levels, while the number of candidate locations for the placement of components in three-dimensional space is bounded by cubic growth. Therefore, the interstage delay must grow between successive stages as the system is scaled up in size. Since number of processors supported by a network with i stages of unit trees is 64^i, the system grows rapidly and will require only a few stages even for very large systems.

4.4 Physical Structure

A natural approach to accommodating this 4 to 1 area convergence (Section 4.3) in a world limited to three dimensions is to build hollow cubes. If we select one face as the "top" of the cube, the four adjacent faces accommodate four times the surface area of the top and naturally four times the number of unit tree stacks of a given size. As such, the "sides" contain the converging stacks from one level, and the "top" contains the unit tree stacks at the next level up the tree to which the "side" stacks are converging. The "bottom" remains open, free of stacks. The "bottom" could be used to shorten wires slightly, but utilizing it in that manner would decrease accessibility to the cube's interior. Figures 7 and 8 show this basic hollow cube structure.

At the next stage of convergence, the hollow cube unit just described is treated as the base unit. Each such unit can be arranged so that its "top" is treated as the basic unit structure for the next stage. We then arrange 16 of these basic units into a square with 4 basic units on each side. A plane of unit trees of size equal to each of the sides is then used for the "top". The next hollow cube stage is shown in Figure 9 as the natural extension of Figure 7. This progression can be



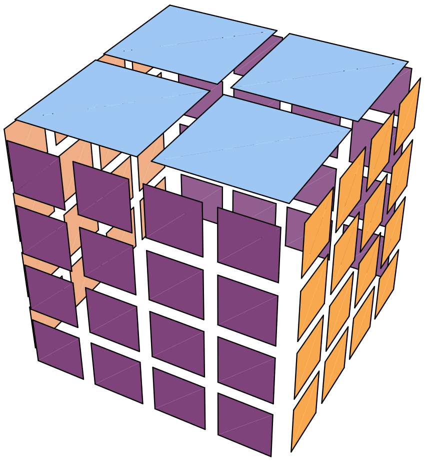


Figure 7: First Level Hollow Cube Geometry (left)

Figure 8: Hollow Cube with Top and Side Stacks of Different Sizes (right)

continued in a recursive manner *ad infinitum* allowing arbitrarily large fat-tree networks to be constructed.

4.5 Features

The hollow cube configuration seems practical while giving reasonable performance for the sizes of interest during the next decade. It is not known to be optimal for minimizing the interstage wiring distances. Finding an optimal solution which retains sufficient accessibility is still an open issue. While this hollow cube structure may not be the most compact structure, it does exhibit a number of desirable properties.

When interconnecting unit tree stack stages, physical channels out of the top of one set of unit trees connects to channels out of the bottom of another set. The channel capacity interconnecting the stages of stacks is thus largely surface area limited. The hollow cube structure allows maximum exposure between the surfaces that need to be connected.

Since the structure is "hollow," connections can be wired through the free space in the center, making the maximum wire length only $\sqrt{3}$ times the length of a side. Every time the number of processors increases by a factor of 64, the maximum length increases by roughly a factor of four due to the factor of four growth in side size associated with each additional level of the hollow cube.[3]

In the hollow cube geometry, individual stacks are reasonably accessible for repair. Since the cubes are hollow, it is possible to access any individual stack without moving any other stack. This accessibility allows repair and inspection

[3] As this progression continues, it becomes necessary to take the size of the "sides" of each square into account, making the increase slightly more than a factor of four. The additional size, though, is only a second order effect.

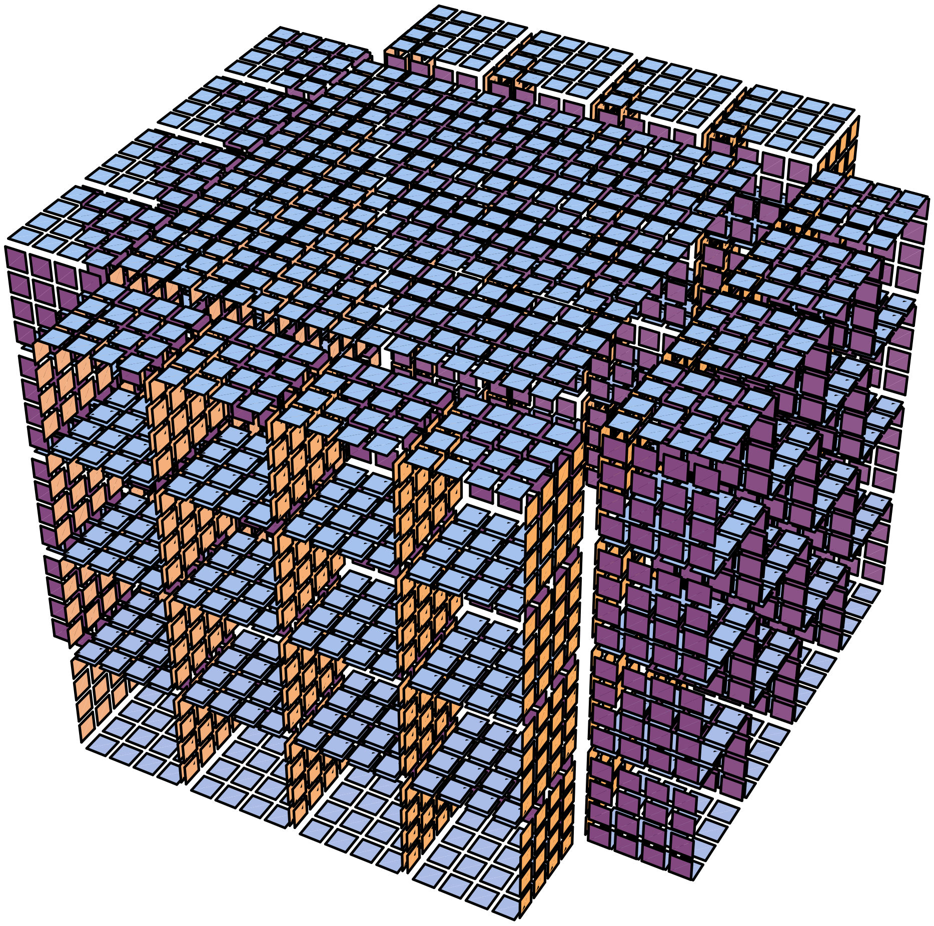


Figure 9: Second Level Hollow Cube Geometry

without interfering with the bulk of the network operation. The numerous wires inside the cube may decrease accessibility, but that effect can be minimized using optical interconnection (*e.g.* [11] [1]) across the free-space in the center of the cube. The "missing" sixth wall of the cube can be used to allow entrance into the center of the structure for maintenance and repair.

5 Hybrid Fat-Trees

Fat-trees allow us to exploit a considerable amount of locality at the expense of lengthening the paths between some processors. Bidelta networks fall at the opposite extreme of the locality spectrum where everything is uniformly close or distant. Another interesting structure to consider is a *hybrid fat-tree*. A hybrid fat-tree is a compromise between the close uniform connections in the bidelta network and the locality and scalability of the fat-tree network. In a hybrid fat-tree, the main tree structure is constructed exactly as described in previous

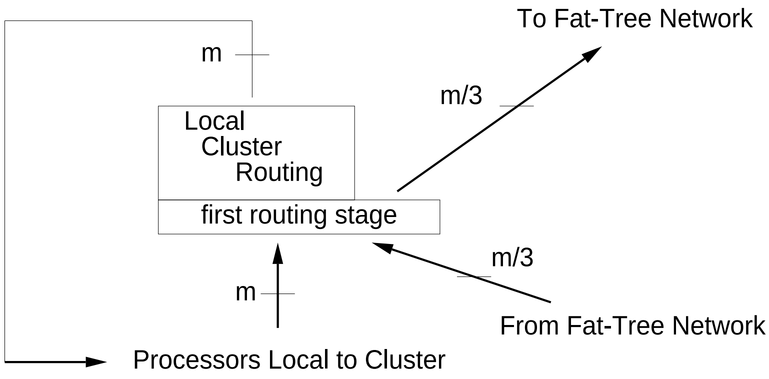


Figure 10: Bidelta Cluster at Leaves of Fat-Tree

sections. However, the leaves of the hybrid fat-tree are themselves small bidelta networks instead of individual processing nodes. With small bidelta networks forming the leaves of the hybrid fat-tree, small to moderate clusters of processors can efficiently work closely together while still retaining reasonable ability to communicate with the rest of the network.

5.1 Bidelta Leaf Clusters

The only additional building block needed to build hybrid fat-trees is the *bidelta leaf cluster* stack. A bidelta leaf stack is constructed in much the same way as a Transit bidelta stack [5]. A bidelta leaf stack is composed of several stages of switching. Each stage switches among four logical directions. The first stage is unique in that only three of the four logical directions through the first stage route to routers in the next stage of the stack. The fourth logical direction through the first routing stage connects to the fat-tree network. The remaining stages in the bidelta leaf stack perform routing purely within the leaf cluster. To allow connections into the leaf cluster from the fat-tree portion of the network, one-fourth of the inputs to the first routing stage come from the fat-tree network rather than from the leaf cluster processing nodes. Figure 10 shows a diagram of a bidelta leaf cluster. Bidelta leaf clusters which support 12, 48, and 192 processing nodes can be easily constructed using RN1 and the current packaging technology (Section 2); these bidelta leaf stacks are referred to as B_{12}, B_{48}, and B_{192}, respectively.

5.2 Example Hybrid Fat-Trees

Hybrid fat-trees are constructed identically to standard fat-trees. Bidelta leaf clusters appear at the lowest level of the tree instead of $UT_{64\times 2}$ unit trees. $UT_{64\times 8}$ unit trees form the inner tree nodes. The hollow cube geometry (Section 4) accommodates the bidelta leaves nicely.

12K A 12,288 processor interconnection network can be built with B_{192} leaf clusters and one stage of $UT_{64\times8}$ unit trees. Since a B_{192} cluster supports 192 processors, we need 64 B_{192} stacks at the leaves of the hybrid fat-tree. The upper tree level can then be built out of 16 $UT_{64\times8}$ unit trees. The resulting hollow cube structure looks much like the one shown in Figure 7.

768K Adding a second level of unit trees to the previous example allows us to connect $64^2 \cdot 192$ =786,432 processors. This requires 4096 bidelta leaf clusters since one B_{192} is needed for each set of 192 processors. The lowest level of unit trees is composed of 1024 $UT_{64\times8}$ unit trees since, on average, one $UT_{64\times8}$ unit tree is needed for every four B_{192} leaf clusters. Finally, 256 $UT_{64\times8}$ unit trees are needed to form the top level of unit trees since the channel capacity out of the top of a unit tree is always one-fourth the total capacity into its bottom. Figure 9 depicts the basic hollow-cube structure for a network of this size.

6 Performance

6.1 Latency

Latency in the unit tree based network is superior to the latency of a naive implementation of a fat-tree network in a number of ways. Skipping tree stages on the route up the tree reduces the number of stages of routing incurred while traveling toward the root by roughly a factor of three. Utilizing the RN1 component, which is a constant sized switch, keeps the routing delay at each stage constant at the cost of incomplete concentration. Schemes described in [2] allow the clock cycle for switching to run at a rate independent of the signal delays incurred while crossing long wires. A long wire only affects the latency of a connection which actually traverses it.

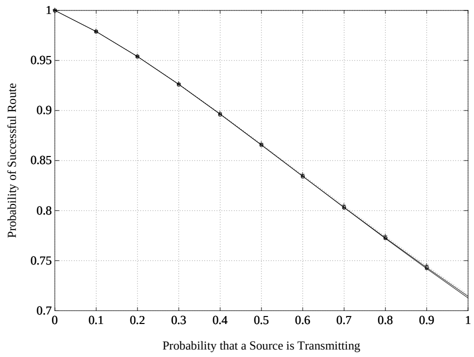
Using the RN1 routing component running at 100 MHz and suitable technology assumptions, Table 3 shows the latency for two representative fat-tree networks. The latency given in Table 3 represents the time required to send data through the network from one endpoint to another assuming the connection is made successfully. [2] provides the derivation of these latency figures, including the assumptions made. Section 6.3 below describes the probability of successfully opening a connection through the network.

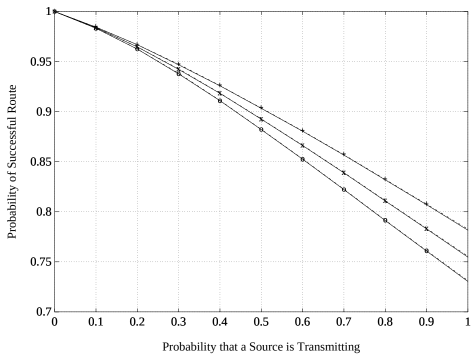
6.2 Locality

When building large networks supporting on the order of thousands to millions of processing nodes, it is clear that locality must be exploited to obtain reasonable performance. The fat-trees and hybrid fat-trees described in this paper each have a natural architectural locality. Tables 4 through 7 summarize the locality structure for several of these fat-tree structures. The relative merit of each of these network architectures is highly dependent on the corresponding locality structure which can be exploited in the communications which utilize the network. In order to

Network Size and Organization	Latency Worst Case	Latency Best Case
Full Fat-Tree 262,144 processors	230 ns	30 ns
Hybrid Fat-Tree 786,432 processors	320 ns	40 ns

Table 3: Network Latency Examples





Curve	Leaf Type
Top	B_{12}
Middle	B_{48}
Bottom	B_{192}

Figure 11: Normalized Routing Statistics for Full Fat-Trees(left)

Figure 12: Normalized Routing Statistics for Hybrid Fat-Tree(right)

achieve optimal performance, the communication patterns in programs should exhibit locality comparable to the architectural locality of the network.

6.3 Routing Performance

With locality comparable to that shown in Tables 5 through 7, the routing performance of these fat-tree structures is favorable to comparably sized flat bidelta networks [2]. The probability of obtaining a successful route in a fully loaded network is in the 70% to 80% range even for networks with three-quarters of a million processors. Figures 11 and 12 summarize the normalized probability of achieving a successful route through the network as a function of network loading. These routing statistics are based on the analytical modeling techniques of [6].

Legend for Tables 5 through 7

n	–	level of fat-tree
$P_{any}(n)$	–	probability that a connection is routed through tree level n
$P_{part}(n)$	–	probability that a connection is routed to a particular processor through tree level n

Table 4: Locality Legend

n	**Processors Supported**					
	64		4096		262144	
	$P_{any}(n)$	$P_{part}(n)$	$P_{any}(n)$	$P_{part}(n)$	$P_{any}(n)$	$P_{part}(n)$
1	0.333	0.111	0.278	9.27×10^{-2}	0.274	9.13×10^{-2}
2	0.333	2.78×10^{-2}	0.278	2.32×10^{-2}	0.274	2.28×10^{-2}
3	0.333	6.94×10^{-3}	0.278	5.79×10^{-3}	0.274	5.71×10^{-3}
4			5.53×10^{-2}	2.88×10^{-4}	3.17×10^{-2}	1.65×10^{-4}
5			5.53×10^{-2}	7.20×10^{-5}	3.17×10^{-2}	4.13×10^{-5}
6			5.53×10^{-2}	1.80×10^{-5}	3.17×10^{-2}	1.03×10^{-5}
7					1.06×10^{-2}	8.62×10^{-7}
8					1.06×10^{-2}	2.16×10^{-7}
9					1.06×10^{-2}	5.39×10^{-8}

Table 5: Locality Structure of Full Fat-Trees

n	$P_{any}(n)$	$P_{part}(n)$		
		B_{12}	B_{48}	B_{12}
0 (leaf)	0.75	6.82×10^{-2}	1.60×10^{-2}	3.93×10^{-3}
1	8.33×10^{-2}	2.31×10^{-3}	5.78×10^{-4}	1.45×10^{-4}
2	8.33×10^{-2}	5.78×10^{-4}	1.45×10^{-4}	3.62×10^{-5}
3	8.33×10^{-2}	1.45×10^{-4}	3.62×10^{-5}	9.04×10^{-6}

Table 6: Locality Structure of Hybrid Fat-Trees (Single Unit Tree Stage)

n	$P_{any}(n)$	$P_{part}(n)$		
		B_{12}	B_{48}	B_{12}
0 (leaf)	0.75	6.82×10^{-2}	1.60×10^{-2}	3.93×10^{-3}
1	6.95×10^{-2}	1.93×10^{-3}	4.83×10^{-4}	1.21×10^{-4}
2	6.95×10^{-2}	4.83×10^{-4}	1.21×10^{-4}	3.02×10^{-5}
3	6.95×10^{-2}	1.21×10^{-4}	3.02×10^{-5}	7.54×10^{-6}
4	6.95×10^{-2}	3.02×10^{-5}	7.54×10^{-6}	1.89×10^{-6}
5	6.95×10^{-2}	7.54×10^{-6}	1.89×10^{-6}	4.71×10^{-7}
6	6.95×10^{-2}	1.89×10^{-6}	4.71×10^{-7}	1.18×10^{-7}

Table 7: Locality Structure of Full Fat-Trees (Two Unit Tree Stages)

Acknowledgments

Tom Knight originally suggested the idea of using fat-tree structures as a means for scaling Transit networks. Tom Knight and Henry Minsky were invaluable for discussion and criticism of these ideas throughout the development process. Discussions with Tom Leighton, Charles Leiserson, and Alex Ishii were valuable to my understanding of the more theoretical aspects of network organization. Pat Sobalvarro was helpful in providing the basic tools for the probabilistic analysis. Fred Drenckhahn is responsible for fleshing out many of the ideas in the Transit packaging scheme. Thanks to Tom Knight, Mike Bolotski, and Ellen Spertus for commenting on early versions of this paper.

This research is supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-0825.

References

[1] L. A. Bergman, A. R. Johnston, R. Nixon, S. C. Esener, C. C. Guest, P. K. Yu, T. J. Drabik, and M. R. Feldman S. H. Lee. Holographic optical interconnects for vlsi. *Optical Engineering*, 25(10):1009–1118, October 1986.

[2] André DeHon. Fat-tree routing for transit. AI Technical Report 1224, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, April 1990.

[3] André DeHon, Thomas F. Knight Jr., and Henry Minsky. Fault-tolerant design for multistage routing networks. AI memo 1225, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, April 1990.

[4] Ronald I. Greenberg and Charles E. Leiserson. Randomized routing on fat-trees. In *IEEE 26th Annual Symposium on the Foundations of Computer Science*. IEEE, November 1985.

[5] Thomas F. Knight Jr. Technologies for low-latency interconnection switches. In *Symposium on parallel architecures and algorithms*. ACM, June 1989.

[6] Thomas F. Knight Jr. and Patrick G. Sobalvarro. Routing statistics for unqueued banyan networks. AI memo 1101, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, September 1990.

[7] F. T. Leighton. The role of randomness in the design of parallel architectures. In William J. Dally, editor, *Advanced Research in VLSI*. MIT Press, 1990. Proceedings of the Sixth MIT Conference.

[8] Tom Leighton and Bruce Maggs. Expanders might be practical: fast algorithms for routing around faults on multibutterflies. In *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989.

[9] Charles E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE transactions on Computers*, C-34(10):892–901, October 1985.

[10] Henry Q. Minsky. Rn1 data router. Transit Note 26, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, July 1990.

[11] W. H. Wu, L. A. Bergman, A. R. Johnston, C. C. Guest, S. C. Esener, P. K. Yo, M. R. Feldman, and S. H. Lee. Implementation of optical interconnections for vlsi. *IEEE Transactions on Electron Devices*, 34(3):706–714, March 1987.