Performance, Cost, and Energy Evaluation of Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network [*]

Hiroki Matsutani[1], Michihiro Koibuchi[2], and Hideharu Amano[1]

[1]Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama,
JAPAN 223-8522
{matutani,hunga}@am.ics.keio.ac.jp

[2]National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo,
JAPAN 101-8430
koibuchi@nii.ac.jp

Abstract

Fat H-Tree is a novel tree-based interconnection network providing a torus structure, which is formed by combining two folded H-Tree networks, and is an attractive alternative to tree-based networks such as Fat Trees in a microarchitecture domain. In this paper, we introduce Fat H-Tree and its deadlock-free routing algorithms. The performance of Fat H-Tree is evaluated using real application traces, and the result is compared with those of other tree-based networks. The network logic area and wire resources for Fat H-Tree are computed based on a typical implementation of on-chip routers using a 0.18μm standard cell library. In addition, the energy consumption is estimated based on the gate-level power analysis. The results show that 1) Fat H-Tree outperforms Fat Tree with two upward and four downward connections in terms of throughput and average hop count; 2) Fat H-Tree requires 19.3%-26.4% smaller network logic area compared with the Fat Tree; 3) Fat H-Tree consumes 8.3%-8.6% less energy compared with the Fat Tree due to its short average hop count; 4) Fat H-Tree uses slightly more wire resources compared with the Fat Tree, but the current process technology can provide sufficient wire resources for implementing Fat H-Tree based on-chip networks.

1 Introduction

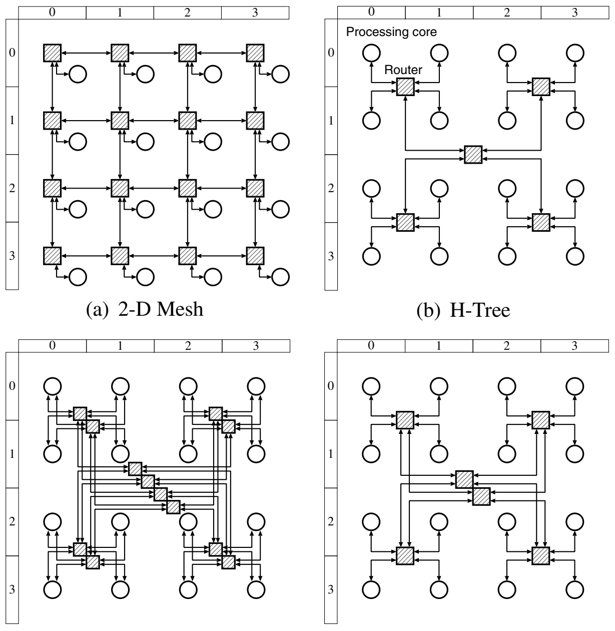
The advance of the semiconductor technology allows us to integrate a number of processing cores on a single chip, and various types of Network-on-Chip (NoC) have been studied to connect them by introducing a network structure similar to that in parallel computers[3, 4].

[*]This work was supported by Joint Research Fund, "Network-on-Chip Architecture," National Institute of Informatics.

NoCs have been utilized not only for high-performance microarchitectures but also for cost-effective embedded devices mostly used in consumer equipment such as set-top boxes or mobile wireless devices. Such embedded applications often demand very tight design constraints in terms of cost and performance; thus the silicon budget available for their on-chip network infrastructure should be modest. On the other hand, NoCs are able to exploit the enormous wire resources, unlike inter-chip interconnects whose bandwidth is usually limited by the pin-count limitation problems outside the chip. Assuming a 0.1μm CMOS technology with 0.5μm minimum wire pitch, for example, a 3mm $\times$ 3mm tile can exploit up to 6,000 wires on each metal layer as illustrated in [4]. Finding the on-chip networks that effectively use large numbers of wires for low latency and high throughput communication with a modest silicon budget is thus essential for rapidly evolving embedded devices.

Two-dimensional mesh and torus[4] have been employed as a typical on-chip interconnect, because their grid-based regular arrangement is intuitively considered to be matched to the two-dimensional VLSI layout. On the other hand, constant attention has been focused on tree-based topologies, because of their relatively short hop-count that enables lower latency communication compared with mesh or torus. In the case of tree-based networks as well as grid-based ones, their performance and area cost have been widely studied [6, 8, 9], and it has been observed that a tree-based network achieves at least as high throughput as a mesh for equivalent chip sizes. In addition, since various NoCs use a tree-based structure[1], we mainly focus on trees for cost-efficient on-chip networks. Note that a comprehensive comparison of tree and grid structures is beyond the scope of our paper.

Figure 1 shows typical on-chip tree-based topologies with low node degree, where a white circle represents a processing core and a shaded square represents a router con-



(a) 2-D Mesh (b) H-Tree

(c) (2,4,2) Fat Tree (d) (2,4,1) Fat Tree

Figure 1. Typical interconnects and their two-dimensional layout

necting other routers or cores. They have different numbers of routers, different link lengths, and different numbers of ports per router, all of which affect throughput, amount of hardware for network resources, and energy consumption.

H-Tree has one upward and four downward connections. Although H-Tree would be placed on a square die of a VLSI chip, the topology is equivalent to a simple tree, so it still has a common weak point of a tree. That is, links or routers around the root of the tree are frequently congested.

To mitigate the congestion around the root of the tree, Fat Tree enhances the number of connections toward the root[9]. As stylized in [9], various forms of Fat Tree can be created, and they can be expressed with a tuple (p, q, c), where p is the number of upward connections, q is the number of downward connections, and c is the number of upward connections that each core has. Figure 1(c) shows a typical $(2, 4, 2)$ Fat Tree, in which each router (except for top-rank routers) has two upward and four downward connections, and each core has two upward connections. This is a network architecture employed in CM-5[10]. On the other hand, Figure 1(d) shows a more reasonable one labeled with $(2, 4, 1)$, which means every core has only one upward connection. In this paper, we consider $(2, 4, 1)$ and $(2, 4, 2)$ to be typical on-chip Fat Trees with low node degree for comparison purposes.

We proposed a novel tree-based interconnection network called Fat H-Tree that is unlike the existing interconnects mentioned above and is an attractive alternative to the other tree-based topologies. A Fat H-Tree has a torus structure, which is formed by only combining two H-Trees, and it achieves as high performance as the torus by using a smaller network logic. Interconnection networks that have both tree and grid structures have been researched for large-scale parallel machines; for example, Recursive Diagonal Torus (RDT)[14] is an extended hierarchical torus which also has tree properties. However, since RDT was originally designed for massively parallel machines, its node degree is high (e.g., at least 8), so its connection structure tends to be costly in a microarchitecture domain and its layout on a chip is also difficult.

Although we first presented the idea of Fat H-Tree a couple of years ago[13], we did not do detailed evaluations of its performance, cost, and energy consumption. In this paper, first, we revise the description of the Fat H-Tree and present three routing algorithms for different purposes. Second, we evaluate the performance of Fat H-Tree through the flit-level simulation. Third, the required hardware amount and wire resources are computed based on a typical implementation of NoC routers using a 0.18μm standard cell library. In addition, the energy consumption is estimated based on the gate-level power analysis of them.

In this paper, Section 2 and 3 introduce Fat H-Tree and its deadlock-free routing algorithms. Section 4 discusses the cost and performance advantages of Fat H-Tree over typical tree-based networks, based on the topological properties. The theory is demonstrated through simulations in Section 5. Section 6 concludes this paper.

2 Fat H-Tree

Fat H-Tree is a novel tree-based interconnection network providing a torus structure, which is formed by combining two H-Tree networks, called **red tree** and **black tree**. Similar to the (2,4,2) Fat Tree in Figure 1(c), every processing core in a Fat H-Tree has two ports: one for connecting to the red tree and the other one for the black tree. The network interface (NI) in the Fat H-Tree has a routing function to forward packets from red to black and vice versa. This function provides torus-like alternative paths in the Fat H-Tree, which greatly improve its performance (see Section 5.4).

a) Red Tree Figure 2 shows the red tree, where the number in a router (e.g., 1,2, or 3) represents its rank. Assume that $4^n = 2^{2n}$ cores are aligned in a $2^n \times 2^n$ two-dimensional grid square, and two-dimensional coordinates (x, y) are assigned to each core. We call such a core a rank-0 router from the network point of view. For a rank-0 router (x, y), the red-tree coordinates $R(r_0, r_1, ...r_{n-1})$ are

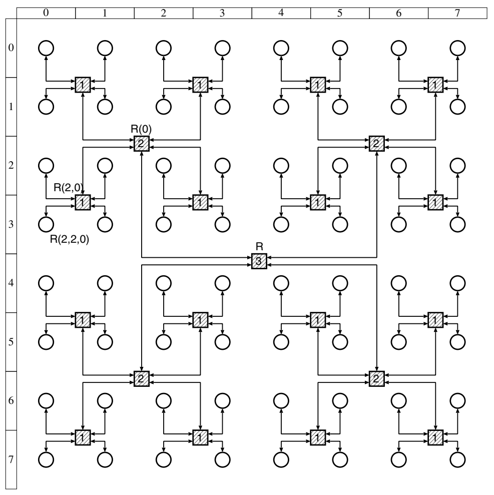


Figure 2. Red Tree

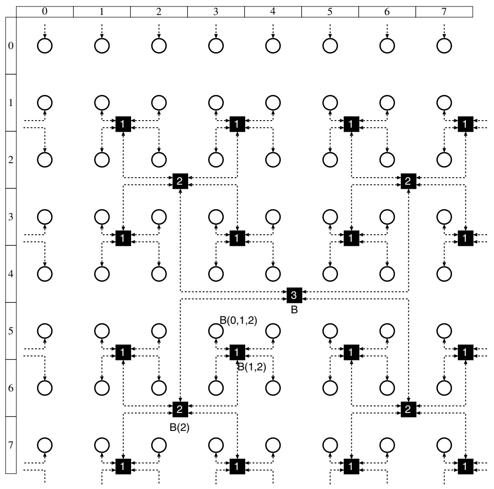


Figure 3. Black Tree

assigned as follows.

$$r_i = ((x/2^i) \bmod 2) + 2 \times ((y/2^i) \bmod 2) \qquad (1)$$

For each i from 0 to $n-1$, four rank-i red routers $R(r_i, ...r_{n-1})$ which have the same part of coordinates $R(r_{i+1}, ...r_{n-1})$ are connected to the rank-($i+1$) red router labeled with $R(r_{i+1}, ...r_{n-1})$. The top-rank router in the red tree has thus coordinates R. Figure 2 shows R, $R(0)$, $R(2,0)$, and $R(2,2,0)$ as examples of red-tree coordinates.

b) Black Tree Figure 3 shows the black tree, which is located to the lower right of the red tree. For a rank-0 router (x, y), the black-tree coordinates $B(b_0, b_1, ...b_{n-1})$ are assigned as follows.

$$b_i = ((((x-1) \bmod 2^n)/2^i) \bmod 2) + \\ 2 \times ((((y-1) \bmod 2^n)/2^i) \bmod 2) \qquad (2)$$

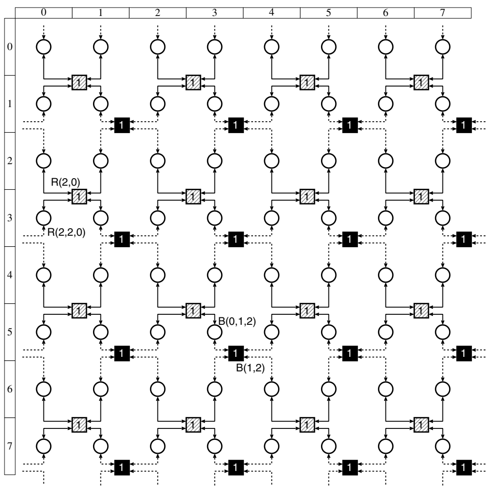


Figure 4. Fat H-Tree (rank-2 or upper routers are not shown)

For each i from 0 to $n-1$, four rank-i black routers $B(b_i, ...b_{n-1})$ which have the same part of coordinates $B(b_{i+1}, ...b_{n-1})$ are connected to the rank-($i+1$) black router labeled with $B(b_{i+1}, ...b_{n-1})$. Figure 3 shows B, $B(2)$, $B(1,2)$, and $B(0,1,2)$ as examples of black-tree coordinates.

c) Fat H-Tree On $2^n \times 2^n$ rank-0 routers, a Fat H-Tree is formed with an n-rank red tree and an n-rank black tree. As shown in Figure 4, the Fat H-Tree has a torus structure, which is formed with rank-0 and rank-1 routers in both trees. Note that the rank-2 or upper routers in the Fat H-Tree are omitted in the figure for ease of understanding.

d) Two-Dimensional Layout A Fat H-Tree can be folded to avoid long feedback links laid across the chip (e.g., links connecting the rightmost/top router and the leftmost/bottom router). In the same manner as a folded two-dimensional torus, a Fat H-Tree can be folded. As shown in Figure 5, the order of nodes is changed so that every link is connected to the next neighboring node. Obviously, Figures 4 and 5 are topologically equivalent. The placement of routers and links is well distributed as shown in Figure 5.

3 Routing Algorithms

Packet routing is a crucial factor to make the best use of network resources on a Fat H-Tree. We propose three deadlock-free routing algorithms for partitioning into two sub-networks, tree-based paths, and torus-based paths.

1. **Single Tree Routing (STR) :**
   Either the red or the black tree is selected on a per-packet basis. That is, a single tree is selected at the

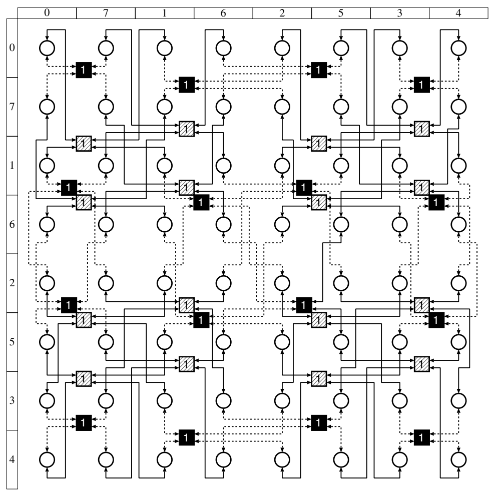


Figure 5. Folded Fat H-Tree (rank-2 or upper routers are not shown)

source node prior to packet injection, and this selection cannot be changed at intermediate routers.

2. **Dual Tree Routing (DTR) :**
   Unlike STR, DTR allows transitions between trees at intermediate rank-0 routers in order to always transfer packets on minimal paths. To remove cyclic dependencies between trees, we use virtual channels with the following rule: starting from the virtual-channel number $zero$, the current virtual-channel number is increased only when a packet is forwarded from red to black at a rank-0 intermediate router. This requires $\lfloor H_{max}/4 \rfloor + 1$ virtual channels, because the longest path changes trees from red to black up to $\lfloor H_{max}/4 \rfloor$ times, where H_{max} is the maximum hop count of the routing.

3. **Torus Routing (TOR) :**
   TOR also allows transitions between trees, but uses only a torus structure formed with rank-0 and rank-1 routers, as illustrated in Figure 4. Hence, it is possible to take non-minimal paths. TOR thus requires $\lfloor H_{max}/4 \rfloor + 1$ virtual channels, where H_{max} is the maximum hop count of the routing.

Theorem *STR, DTR, and TOR are deadlock-free under the condition that $zero$, $\lfloor H_{max}/4 \rfloor + 1$, and $\lfloor H_{max}/4 \rfloor + 1$ virtual channels are provided, where H_{max} is the maximum hop count of each routing algorithm, respectively.*

Proof *Since STR sends packets along a tree which is acyclic, STR guarantees deadlock-freedom. DTR and TOR are also deadlock-free because no cyclic dependency occurs as follows.*

1. *No cyclic dependency is formed within each tree.*
2. *No cyclic dependency is formed across trees, because packets are passed from the red to the black tree by virtual-channel transition in increasing order.* ∎

Although STR has the disadvantage of non-minimal paths, it can physically partition the network resources into two sub-networks, both of which provide connectivity between all pairs of processing cores, for different purposes (e.g., pre-scheduled and dynamic networks). It can also be used for fault-tolerance.

DTR is a minimal routing, and it usually offers high throughput and low latency. TOR is completely free from a tree's weak point (i.e., root bottleneck), but its average hop count will be increased, because it does not use all the network resources. It thus guarantees connectivity even if rank-i routers fail, where $i > 1$. In addition, by supporting hardware broadcast and/or multicast operation based on a tree, unicasts along TOR and hardware multicasts can be separated on a Fat H-Tree. This routing application presents the possibility of customizing to fit to the traffic locality the application generates. Since there are no routing algorithms that are best performance on any applications, each microarchitecture system should carefully select a routing algorithm. Section 5.4 shows the throughput results of the proposed routing algorithms.

4 Topological Properties

We discuss the cost and performance advantages of Fat H-Tree over typical tree-based networks, based on the topological properties in terms of channel bisection, average hop count, number of routers, and total length of links. The properties were confirmed by conducting simulations, as described in the next section.

4.1 Ideal Throughput

The ideal throughput of a network is the data acceptance rate that would result from perfectly balanced routing and flow control with no idle cycles; it is calculated as [5]

$$\Theta_{ideal} \leq \frac{2bB_c}{N} \tag{3}$$

where N is the number of cores, b is the channel bandwidth, and B_c is the channel bisection of the network.

Table 1 shows the channel bisection of typical networks. In the table, "HT" represents H-Tree, "FT1" is (2,4,1) Fat Tree, "FT2" is (2,4,2) Fat Tree, and "FHT" is Fat H-Tree.

Again, the number of cores is $N = 2^n \times 2^n$; therefore, the number of ranks in an N-core tree is $\log_4(N) = \log_4(4^n) = n$. The channel bisection of an N-core Fat

Table 1. Channel bisection B_c

	N-core	16-core	64-core	256-core
HT	4	4	4	4
FT1	2^{n+1}	8	16	32
FT2	2^{n+2}	16	32	64
FHT	$2^{n+2}+8$	24	40	72
Mesh	2^{n+1}	8	16	32
Torus	2^{n+2}	16	32	64

Table 2. Average hop count H_{ave}

	Routing	16-core	64-core	256-core
HT,FT	tree	3.60	5.43	7.36
FHT	STR	3.20	5.02	6.90
FHT	DTR	3.20	4.84	6.78
FHT	TOR	3.20	5.65	10.83
Mesh	DOR†	4.67	7.33	12.67
Torus	DOR†	4.13	6.06	10.03

† Dimension-order routing[5]

H-Tree is $2^{n+2}+8$, where the second term 8 corresponds to that in two H-Tree networks (i.e., red and black trees) and the first term 2^{n+2} is due to the torus structure of the Fat H-Tree. Compared with the ideal throughput of simply doubled H-Trees, the ideal throughput of a Fat H-Tree is greatly improved by the torus structure. Section 5.4 shows case studies of the application throughput on Fat H-Tree by using a flit-level network simulator.

4.2 Average Hop Count

The number of source-destination pairs in an N-core network is N^2-N; thus average hop count in the network is

$$H_{ave} = \frac{1}{N^2-N} \sum_{x,y \in N} H_{(x,y)} \qquad (4)$$

where $H_{(x,y)}$ is the hop count from core-x to core-y. Table 2 shows the average hop count of typical networks in the case of uniform traffic, in which each source sends equally to each destination. The average hop count depends on whether the routing includes non-minimal paths. In the table, STR and TOR in Fat H-Tree are non-minimal. DTR is a minimal routing and it achieves 7.9%-11.1% shorter average hop count compared with Fat Tree.

4.3 Number of Routers

The number of routers in a chip affects the network logic area and the implementation cost.

Assuming that the number of downward connections q is 4, the number of routers in an n-rank H-Tree, R_{ht}, is

Table 3. Number of routers R

	N-core	16-core	64-core	256-core
HT	$(4^n-1)/3$	5	21	85
FT1	$(4^n-2^n)/2$	6	28	120
FT2	4^n-2^n	12	56	240
FHT	$2(4^n-1)/3$	10	42	170
Mesh	N	16	64	256
Torus	N	16	64	256

$$R_{ht} = (q^n-1) \;/\; (q-1) = (4^n-1) \;/\; 3. \qquad (5)$$

Similarly, the number of routers in a (2,4,1) Fat Tree network, R_{ft1}, is

$$R_{ft1} = (q^n-2^n) \;/\; (q-2) = (4^n-2^n) \;/\; 2. \qquad (6)$$

A (2,4,2) Fat Tree has twice as many routers as are in the (2,4,1) Fat Tree; thus the number of routers in the (2,4,2) Fat Tree is $R_{ft2}=2R_{ft1}$. A Fat H-Tree contains two H-Trees, so the number of routers it has is $R_{fht}=2R_{ht}$.

Table 3 lists the number of routers in typical networks. The number of routers in a Fat H-Tree is smaller than that in the (2,4,2) Fat Tree, yet the Fat H-Tree outperforms Fat Tree in terms of ideal throughput and average hop count, as shown in Section 4.1 and 4.2. To assess the cost and performance advantages of Fat H-Tree, we need to consider the amount of hardware for each router and NI in addition to the number of routers. Actually, Fat H-Tree and (2,4,2) Fat Tree require a 2-port NI per core, while the others use a 1-port NI for each core. In addition, an NI in the Fat H-Tree has a routing function that forwards packets from one port to another, and this function would increase the amount of hardware in each NI. In Section 5.1, we implement an entire Fat H-Tree based NoC, and compare it with other typical networks in term of the network logic area.

4.4 Total Unit-Length of Links

Assuming that the distance between neighboring two cores aligned in a two-dimensional grid square is 1-unit, we define U as the total unit-length of links in a network. For instance, a 16-core H-Tree network shown in Figure 1(b) has 16 1-unit length links and four 2-unit length links, thus its U is 24-unit.

The total unit-length of links in an n-rank H-Tree network, U_{ht}, is

$$U_{ht} = \sum_{i=1}^{n} u_{ht}^{i} \cdot r_{ht}^{i} \qquad (7)$$

where u_{ht}^{i} is the total unit-length of links between a rank-i router and its four child routers, and r_{ht}^{i} is the number of

Table 4. Total unit-length of links U

	N-core	16-core	64-core	256-core
HT	Equation 8	24	112	480
FT1	nN	32	192	1,024
FT2	$2nN$	64	384	2,048
FHT	Equation 11	72	392	1,800
Mesh	$2(N-2^n)$	24	112	480
Torus	$4(N-2^n)$	48	224	960

rank-i routers in the H-Tree. Assuming that the number of cores is $N = 2^n \times 2^n$, $u^i_{ht} = 2^{i+1}$ and $r^i_{ht} = N/4^i$, where $1 \le i \le n$. Thus, Equation 7 can be transformed as follows.

$$U_{ht} = \sum_{i=1}^{n} u^i_{ht} \cdot r^i_{ht} = \sum_{i=1}^{n} 2^{i+1} \cdot \frac{N}{4^i} = 2N\left(\frac{2^n-1}{2^n}\right) \quad (8)$$

Similarly, the total unit-length of links in a (2,4,1) Fat Tree network, U_{ft1}, is

$$U_{ft1} = \sum_{i=1}^{n} u^i_{ft1} \cdot r^i_{ft1} = \sum_{i=1}^{n} 2^{i+1} \cdot \frac{N}{2^{i+1}} = nN. \quad (9)$$

A (2,4,2) Fat Tree has double the number of routers in the (2,4,1) Fat Tree; therefore $U_{ft2} = 2U_{ft1}$.

A Fat H-Tree has two folded H-Tree networks, in which each link, except for the links connecting to the top-rank router, requires twice the wire resources of an ordinary H-Tree. By folding the H-Tree, only the top-rank router and its four child routers can be placed inside a 1-unit $\times$ 1-unit grid square. Therefore, the total unit-length of links in a Fat H-Tree network, U_{fht}, can be expressed as follows.

$$u^i_{fht} = \begin{cases} 2u^i_{ht} & 1 \le i \le n-1 \\ 4 & i = n \end{cases} \quad (10)$$

$$U_{fht} = u^n_{fht} \cdot r^n_{fht} + \sum_{i=1}^{n-1} 2^{i+2} \cdot \frac{2N}{4^i} = 8 + 8N\left(\frac{2^{n-1}-1}{2^{n-1}}\right) \quad (11)$$

Table 4 summarizes the above discussion. As for the mesh and torus, we ignore the links between the core and router, which will slightly increase the total unit-length, for the sake of simplicity. Although a Fat H-Tree uses slightly more wire resources compared with the (2,4,2) Fat Tree in 16- and 64-core networks, the impact on the chip design is considered to be modest, because enormous wire resources are available in an NoC, thanks to the current CMOS technology that has six or more metal layers. In Section 5.2, we investigate the impact of wire demand on Fat H-Tree in a 0.18μm CMOS technology.

The longest link in a network affects the wiring delay and the number of repeaters required for wires. As for the Fat H-Tree, each link, except for the links connecting to the top-rank router, requires twice the wire length of a same-sized H-Tree, while the length of the top-rank link is 1-unit because of its folded layout mentioned above. Thus, the longest link length in a Fat H-Tree is the same as those in the H-Tree and Fat Tree.

5 Evaluations

The advantages and disadvantages of Fat H-Tree over Fat Tree were demonstrated through simulations evaluating the network logic area, wire resources, energy consumption, and throughput.

5.1 Hardware Amount

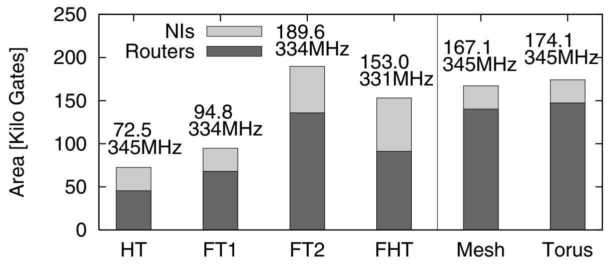
The network logic area in an NoC is mainly composed of routers and network interfaces (NIs) that connect a processing core to a network. Here, Fat H-Tree is compared with other typical networks in terms of network logic area.

We implemented a wormhole router that supports various node degrees. We also developed an NoC generator that automatically connects the routers and NIs in the arbitrary network topologies. Using the Synopsys Design Compiler X-2005.09, we synthesized the generated NoC design with a TSMC 0.18μm standard cell library and estimated the network logic area. The behavior of the synthesized NoC design was confirmed through a gate-level simulation assuming an operating frequency of 250MHz.

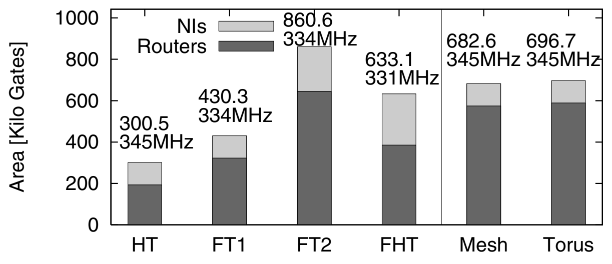
The router architecture was fully pipelined, and it transferred a header flit through four pipeline stages that consisted of a routing computation, virtual-channel allocation, crossbar allocation, and crossbar traversal. The flit-width was set to 32 bits, and each pipeline stage had a buffer for storing one flit. The routing decisions were stored in the header flit prior to packet injection (i.e., source routing); thus routing tables that require register files for storing routing paths were not needed in each router, resulting a low cost router implementation.

The NI has to be designed to interface between a processing core and a network with a minimum hardware amount. We implemented a simple NI that employs a 2-flit FIFO buffer for both the core-to-network and network-to-core interfaces. Each core in a Fat H-Tree and a (2,4,2) Fat Tree has two upward connections, while the other networks have only one connection. For the Fat H-Tree and the (2,4,2) Fat Tree, we also implemented a 2-port NI that had two sets of FIFO buffers. In addition, an NI of Fat H-Tree has to be designed to support a routing function to forward packets from one port to another. This function requires a small 2-input multiplexer for each of the four output ports in the NI.

Figure 6 shows the synthesis results of typical 16- and 64-core networks. Although Fat H-Tree requires the largest



(a) 16-core network



(b) 64-core network

Figure 6. Network logic area

NI area because of its 2-port interfaces and packet forwarding function, the number of routers in a Fat H-Tree is relatively small, as shown in Section 4.3. As a result, Fat H-Tree is smaller than 2-D mesh, torus, and (2,4,2) Fat Tree. In particular, Fat H-Tree consumes 19.3%-26.4% smaller network logic area compared with the (2,4,2) Fat Tree. As for the operating frequency plots, little difference exists among them, as shown in Figure 6.

5.2 Wire Resources

As shown in Section 4.4, Fat H-Tree slightly stretches the total unit-length of links compared with the (2,4,2) Fat Tree. However, the impact is considered to be modest, because the enormous wire resources available in an NoC, thanks to the current process technology that has six or more metal layers. In this section, we first estimate the required wire length for a Fat H-Tree, and then we calculate the utilization ratio of the entire wire resources in a chip to demonstrate the impact of wire demand on Fat H-Tree.

The required wire length for a network is estimated as

$$W = 2Uwl \tag{12}$$

where U is the total unit-length of links in the network, l is the distance between neighboring two cores (e.g., 1.5mm), and w is the bit-width of a channel. Assuming a 12mm $\times$ 12mm chip, the distance between neighboring two cores, l, is 3.0mm for a 16-core network and 1.5mm for a 64-core network. Figure 7 shows the required wire length in the cases of 16- and 64-core typical networks.

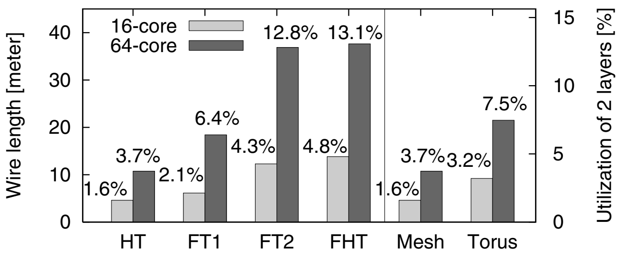


Figure 7. Required wire length W and utilization ratio of two metal layers

We now shift to the utilization ratio of the wire resources available in a 0.18μm CMOS technology. Although the process technology offers six metal layers, we assume that the on-chip network infrastructure can use only two metal layers, while the application logic exploits all layers. Assuming that the minimum wire pitch is 1.0μm in the technology, the 12mm $\times$ 12mm chip has up to 12,000 wires on each metal layer crossing each edge of the chip; thus up to 288m of wires would be available in two metal layers in the chip. Although this assumption often overestimates the entire resources in a chip, it enables us to approximate the wire utilization in a simple way.

Figure 7 also shows the utilization ratio of wires available in two metal layers. Fat H-Tree utilizes 4.8% of wires in the case of a 16-core network, and it uses only 13.1% of wires even in a large 64-core network. This also shows that the differences between Fat H-Tree and (2,4,2) Fat Tree are very small (i.e., 0.3%-0.5%). In addition, the 0.18μm process used in this section is not state-of-the-art technology. The latest technologies such as the 90nm or 65nm process can provide many more wires, which would further ease the wiring issue of Fat H-Tree. It is clear from the above discussion that the current process technology can provide sufficient wire resources for implementing Fat H-Tree based on-chip networks.

5.3 Energy Consumption

The average energy consumed to transmit a single flit from source to destination can be estimated as [12]

$$E_{flit} = wH_{ave}(E_{sw} + E_{link}) \tag{13}$$

where w is the flit-width, H_{ave} is the average hop count, E_{sw} is the average energy to switch a 1-bit data inside a router, and E_{link} is the 1-bit energy consumed in a link.

We used the Synopsys Power Compiler to extract E_{sw} of the router synthesized with the 0.18μm standard cell library. The switching activity of the running router was captured through the gate-level simulation of the synthesized router.

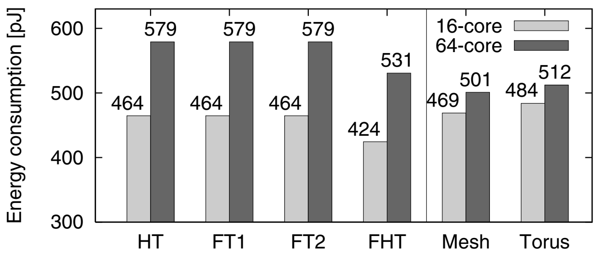


Figure 8. Average energy consumption to transmit a single flit (flit-width = 32-bit)

The gate-level power analysis based on this switching activity shows that E_{sw} is 1.13pJ when the router is operating at 250MHz with a 1.8V supply voltage.

E_{link} can be calculated as

$$E_{link} = dV^2 C_{wire}/2 \qquad (14)$$

where d is the average 1-hop distance (in millimeters), V is the supply voltage, and C_{wire} is the wire capacitance per mm. C_{wire} can be estimated using the method proposed in [7], and is 414fF/mm in the case of a semi-global interconnect in the 0.18μm CMOS technology. For instance, E_{link} is 0.67pJ when the 1-hop distance is 1mm on average.

We assume 32-bit 16- and 64-core networks placed in a 12mm $\times$ 12mm chip, as in Section 5.2. First we estimated the average 1-hop distance, d, using a flit-level simulator, and then we derived E_{flit} based on Equation 13 with the various parameters mentioned above.

Figure 8 shows the results. The energy consumption depends on H_{ave} and d. Fat Trees frequently use the longest links (e.g., 75% of source-destination pairs use the longest ones twice in each flight). On the other hand, Fat H-Tree tries to avoid the congested longest links but the length of each link is longer than those of Fat Trees because of its folded layout. As a result, average 1-hop distance of Fat H-Tree becomes slightly longer than those of Fat Trees. Since the average hop count of Fat H-Tree is shorter than those of Fat Trees and it strongly affects the energy consumption as in Equation 13, Fat H-Tree consumes 8.3%-8.6% less energy compared with Fat Trees.

5.4 Throughput

5.4.1 Simulation Environments

Network Model A flit-level simulator written in C++ was used to confirm deadlock-freedom and measure the throughput on Fat H-Tree. A simple model corresponding to the wormhole router mentioned in Section 5.1 was used as a switching fabric in the simulator. A header flit requires at least three clock cycles to be transferred to the next router or core; one cycle for the routing computation, one cycle for allocating a virtual-channel and a crossbar, and the remaining cycle for transferring the flit to the next router or core. Wormhole switching was used as a switching technique on the router. The nodes inject packets independently of each other, and we set the packet length for 16 flits including one header flit.

Routing Algorithm We used DTR and TOR for Fat H-Tree, and dimension-order routing (DOR) for the 2-D mesh and torus. DTR and TOR in Fat H-Tree and DOR in 2-D torus require virtual channels in order to avoid structural deadlocks; thus we assumed to use two virtual channels for these 16- and 64-core networks. Since the Fat H-Tree and Fat Trees provide alternative paths between source and destination, we employed the path selection algorithm proposed in [11], which selects a path so as to distribute the congestion over the network.

Traffic Pattern As for the traffic pattern, we used uniform synthesis traffic as a baseline. In addition, we used application traces captured from NAS Parallel Benchmark (NPB)[2] programs, because they would enable us to evaluate with various sizes/patterns of real application traffic. We selected five matrix computation programs from NPB: Block Tridiagonal solver (BT), Scalar Pentadiagonal solver (SP), Conjugate Gradient (CG), Multi-Grid solver (MG), and large Integer Sort (IS). The class of problem was set to “W”, and the numbers of tasks was 16 or 64. The task mapping which assigns each task into a core is a crucial factor for the average hop count and performance, and it was optimized for every trace in every network so that the pairs of tasks that transferred a large amount of data could be placed near each other by using Equation 4. We employed the branch-and-bound method as a pruning technique to obtain the optimum mapping within a realistic time frame.

5.4.2 Simulation Results

First we discuss the performance advantages of Fat H-Tree over H-Tree and Fat Tree, and then we experimentally compare Fat H-Tree with 2-D mesh and torus.

Figure 9(a) shows the throughput (accepted traffic) versus the latency with the 16-core uniform traffic on Fat H-Tree and simple trees. The average hop count of each topology is also shown in parentheses. The average hop counts are equal to the ones in Table 2. Fat H-Tree (both cases of FHTD and FHTT) achieves 11.1% shorter average hop count compared with Fat Tree. As for the throughput, Fat H-Tree achieves higher throughput than others. Actually, FHTT outperforms the (2,4,2) Fat Tree by 19.5%.

Similar results are also shown in the NPB traces. Figures 9(b) and 9(c) show the results for 16-core BT and MG traces. Note that we could not show the results of SP, CG, and IS traces because of space limitations. Fat H-Tree outperforms the (2,4,2) Fat Tree in terms of throughput and average hop count. Since the Fat H-Tree requires 19.3%-26.4% smaller network logic area compared with the Fat Tree as reported in Section 5.1, it is clear that it outperforms the Fat Tree in terms of cost-performance. Thus we can see that Fat H-Tree is an attractive alternative to Fat Tree.

Next, we compare FHTD and FHTT. Although both FHTD and FHTT achieve higher throughput than Fat Tree, an interesting tendency can be seen in their performance differences. That is, FHTD outperforms FHTT in the BT.W and SP.W traffic, where most communications are limited between neighboring cores. On the other hand, FHTD is inferior to FHTT in the uniform and IS.W traffic, each of which is dominated by all-to-all communications. FHTD is able to exploit the links around the roots of Fat H-Tree in order to always take the minimal path, and this strategy works well for most real application traffic. However, in the cases of all-to-all communications, FHTD has no other choice to use the links around the roots for taking minimal paths. As a result, FHTD causes congestion around the roots and has slightly lower performance in the cases of all-to-all traffic.

The simulation results with 64-core networks are shown in Figures 9(d)-9(f). As you can see, FHTD and FHTT achieve higher throughput than FT2 in most traces.

We experimentally compared Fat H-Tree with 2-D mesh and torus (Figures 10(a)-10(f)). Fat H-Tree outperforms 2-D mesh in all traces, and it also achieves as high throughput as the torus when a suitable routing algorithm is selected according to the traffic pattern. Since the required silicon budget for a Fat H-Tree is smaller than those for mesh and torus, Fat H-Tree could be comparable to these simple grid-based networks.

6 Conclusions

In this paper, we introduced Fat H-Tree and its deadlock-free routing algorithms. We showed their properties, and evaluated them. The evaluation results provide us with the following conclusions: 1) Fat H-Tree outperforms Fat Tree with two upward and four downward connections in terms of throughput and average hop count; 2) Fat H-Tree requires 19.3%-26.4% smaller network logic area compared with the Fat Tree; 3) Fat H-Tree consumes 8.3%-8.6% less energy compared with the Fat Tree due to its short average hop count; 4) Fat H-Tree uses slightly more wire resources compared with the Fat Tree, but the current process technology can provide sufficient wire resources for implementing Fat H-Tree based NoCs. Although the required silicon budget for a Fat H-Tree is smaller than that for the Fat Tree, Fat H-Tree achieves higher performance compared with the Fat Tree. Thus, Fat H-Tree outperforms the Fat Tree in terms of cost-performance.

Possible improvement of Fat H-Tree is to develop efficient two-dimensional layouts for the general $m \times n$ networks, where $m \neq n$. In addition, we are planning to employ Fat H-Tree as a network architecture of 3-D stacked chips. That is, stacked chips, each of which has its own on-chip H-Tree network, are connected with an inter-chip network. Fat H-Tree will be used to integrate these on-chip and inter-chip networks.

References

[1] A. Andriahantenaina and et. al. SPIN: A Scalable, Packet Switched, On-chip Micro-network. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 70–73, Mar. 2003.

[2] D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A.Woo, and M.Yarrow. The NAS Parallel Benchmarks 2.0. *NAS Technical Report, NAS-95-020*, Dec. 1995.

[3] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, Jan. 2002.

[4] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the Design Automation Conference*, pages 684–689, June 2001.

[5] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[6] A. DeHon. Compact, Multilayer Layout for Butterfly Fat-Tree. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 206–215, July 2000.

[7] R. Ho, K. W. Mai, and M. A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[8] N. Kapre and et. al. Packet-Switched vs. Time-Multiplexed FPGA Overlay Networks. In *Proceedings of the Symposium on Field-Programmable Custom Computing Machines*, pages 205–216, Apr. 2006.

[9] C. E. Leiserson. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, Oct. 1985.

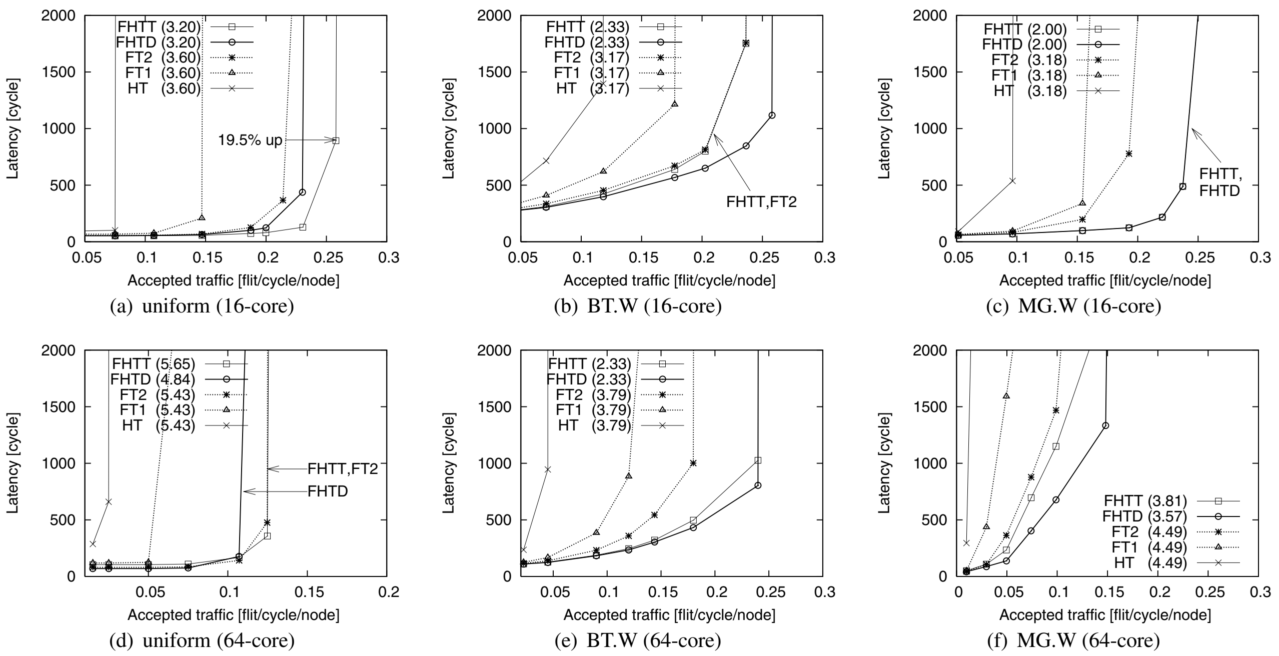
[10] C. E. Leiserson and et. al. The Network Architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, Mar. 1996.

[11] J. C. Sancho and A. Robles. Improving the Up*/Down* Routing Scheme for Networks of Workstations. In *Proceedings of the European Conference on Parallel Computing*, pages 882–889, Aug. 2000.

[12] H. Wang, L.-S. Peh, and S. Malik. A Technology-Aware and Energy-Oriented Topology Exploration for On-Chip Networks. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1238–1243, Mar. 2005.

[13] Y. Yamada, H. Amano, M. Koibuchi, A. Jouraku, K. Anjo, and K. Nishimura. Folded Fat H-Tree: An Interconnection Topology for Dynamically Reconfigurable Processor Array. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, pages 301–311, Aug. 2004.

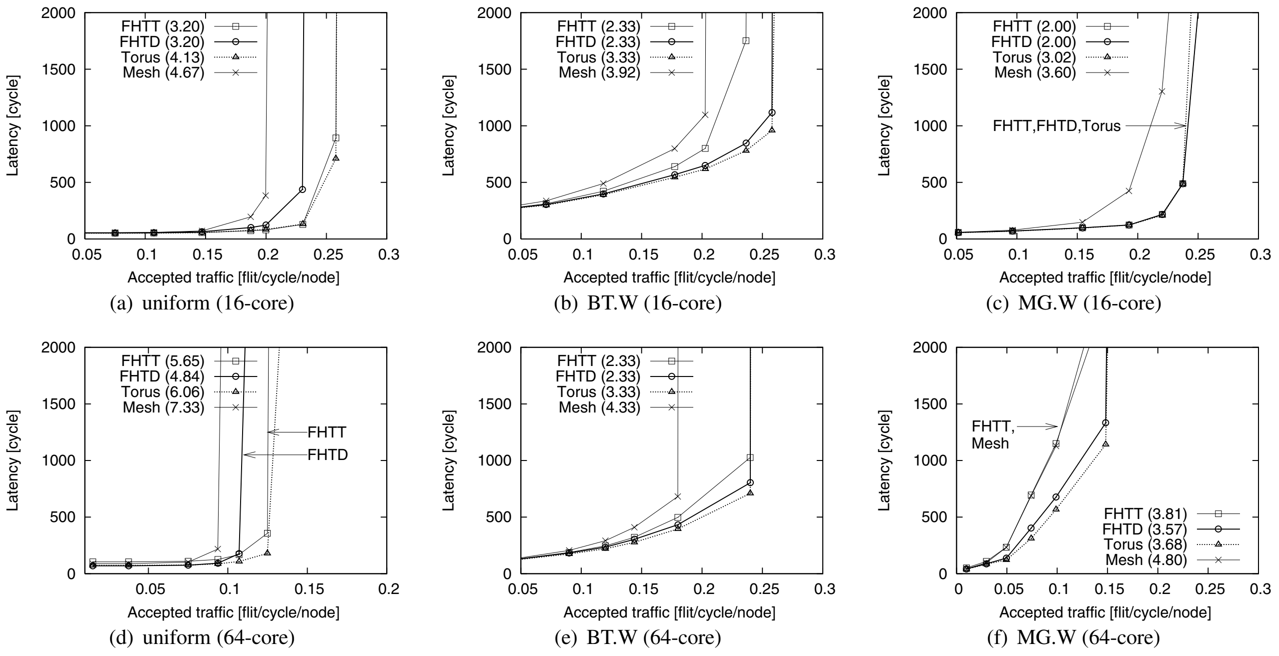
[14] Y. Yang, A. Funahashi, A. Jouraku, H. Nishi, H. Amano, and T. Sueyoshi. Recursive Diagonal Torus: An Interconnection Network for Massively Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 12(7):701–715, July 2001.



(a) uniform (16-core) (b) BT.W (16-core) (c) MG.W (16-core)

(d) uniform (64-core) (e) BT.W (64-core) (f) MG.W (64-core)

Figure 9. Fat H-Tree vs. H-Tree and Fat Trees



(a) uniform (16-core) (b) BT.W (16-core) (c) MG.W (16-core)

(d) uniform (64-core) (e) BT.W (64-core) (f) MG.W (64-core)

Figure 10. Fat H-Tree vs. 2-D mesh and torus