Mesh-of-Trees and Alternative Interconnection Networks for Single-Chip Parallel Processing*
UMIACS-TR 2006-32
(Extended Abstract)†

Aydin O. Balkan Gang Qu Uzi Vishkin
University of Maryland Institute for Advanced Computer Studies (UMIACS)
University of Maryland Department of Electrical and Computer Engineering
{balkanay, gangqu, vishkin}@umd.edu

Abstract

Many applications have stimulated the recent surge of interest single-chip parallel processing. In such machines, it is crucial to implement a high-throughput low-latency interconnection network to connect the on-chip components, especially the processing units and the memory units. In this paper, we propose a new mesh of trees (MoT) implementation of the interconnection network and evaluate it relative to metrics such as wire area, register count, total switch delay, maximum throughput, latency-throughput relation and delay effects of long wires. We show that on-chip interconnection networks can facilitate higher bandwidth between processors and shared first-level cache than previously considered possible. This has significant impact for chip multiprocessing. MoT is also compared, both analytically and experimentally, to some other traditional network topologies, such as hypercube, butterfly, fat trees and butterfly fat trees. When we evaluate a 64-terminal MoT network at 65nm technology, concrete results show that MoT provides higher throughput and lower latency especially when the input traffic (or the on-chip parallelism) is high, at the cost of larger area. A recurring problem in networking and communication is that of achieving good sustained throughput in contrast to just high theoretical peak performance that does not materialize for typical work loads. Our quantitative results demonstrate a clear advantage of the proposed MoT network in the context of single-chip parallel processing.

1 Introduction

The advent of the Billion-transistor chip era coupled with a slow down in clock rate improvement brought about a growing interest in parallel computing. Ongoing expansion in the demands of scientific and commercial computing workloads also contributes to this growth in interests. To date, the outreach of parallel computing has fallen short of historical expectations. This has primarily been attributed to programmability shortcomings of parallel computers. The Parallel Random Access Model (PRAM) is an easy model for parallel algorithmic thinking and for programming. Current multi-chip multiprocessor designs that aim to support the PRAM (such as Tera/Cray MTA [1] and SB-PRAM [2]), although interesting, are constrained by inter-chip interconnections. Latency and bandwidth problems have limited their success in supporting PRAM. With the continuing increase of silicon capacity, it becomes possible to build a single-chip parallel processor as is being demonstrated in the Explicit Multi-Threading (XMT) project [31, 22] that seeks to prototype a *PRAM-On-Chip* vision.

Parallel computing generally requires a larger number of memory accesses than serial computation per clock. A standard technique for hiding access latencies is by feeding functional units with instructions coming from multiple

*Partially supported by grant 0325393 from the National Science Foundation.

†A summary of this paper has been presented in the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2006)

hardware threads. This allows, for example, overlapping several arithmetic instructions as well as read instructions each requiring waiting for data. Such overlap implies a steady and high demand for memory accesses. To facilitate concurrent accesses by many processing elements, memory is normally partitioned on parallel machines [15]. For example, [1] uses 512 memory modules of 128MB each, and [2] uses as many memory modules as processing elements.

To handle the high level of parallelism needed for a PRAM on-a-chip, XMT uses a novel memory architecture where partitioning of data memory starts from the first level of the on-chip cache. It is a challenging task to provide connectivity between many processing units and many memory modules on-chip. A poorly designed interconnection network may create on-chip queuing bottlenecks, when concurrent read and/or write requests are issued to the memory. This will significantly affect the network's throughput, memory latency, and the overall system performance.

1.1 Existing Approaches

Hypercube, butterfly and fat tree topologies and their variations have been popular for interconnection networks in parallel computing studies (e.g. [18, 27, 12, 33, 7]) as well as in machine architectures (e.g. [1, 10, 19]). In all these topologies, data packets between different sources and different destinations can interfere with one another. For background, note also that source to destination traffic is expected to be balanced. This is because that the use of universal hashing [21, 10] will prevent hot spots. On the other hand, *Crossbar* networks provide one standard type of high-throughput interconnection networks, where packets do not interfere. They achieve this by scheduling the switches based on the global state of the network. The overhead for global scheduling may be acceptable with large payloads in messages. However, in the XMT single-chip parallelism context, the messages between processors and cache are very small (one word for *load* instructions and at most two words for the *store* instructions). Therefore, the networks that need to globally schedule the switches will incur significant overheads.

1.2 Contribution and Paper Organization

In this paper, we study the interconnection network design problem for a certain memory architecture that is designed to achieve high single-chip parallelism. We first analyze the existing interconnection network models and conclude that they do not meet the latency and throughput requirements for such interconnection network. We then propose a new approach based on the concept of mesh of trees (MoT). We conduct simulation to evaluate the performance of both the existing approaches and the proposed MoT network.

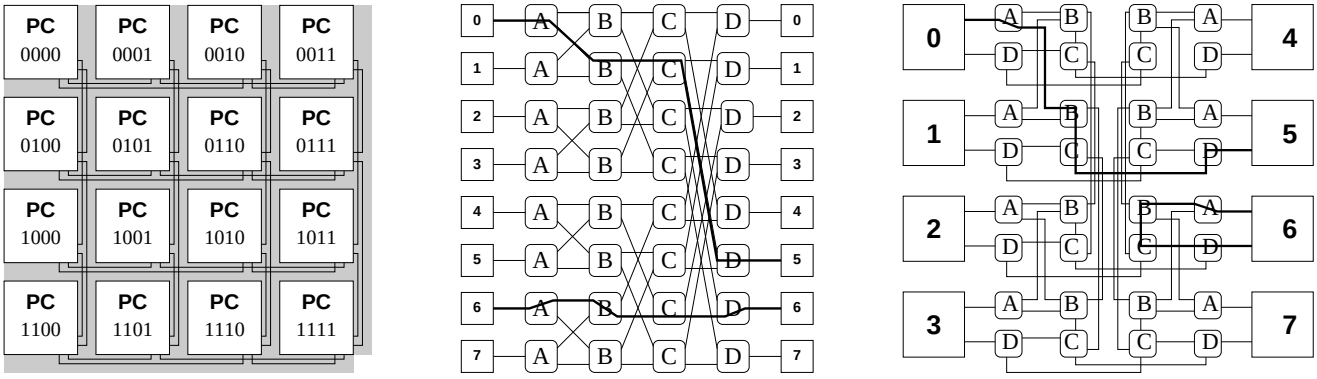
The paper makes two contributions. First, on-chip implementations of several of the most popular interconnection network topologies are compared in terms of total wire area, total switch delay, maximum throughput, and trade-offs between throughput and latency. We are unaware of a similar comprehensive comparative study in the literature. Second, although MoT is a well-known concept for parallel algorithms and architectures, it has not been applied to build interconnection network. We describe the MoT network structure and demonstrate, through both theoretical analysis and simulation, that the proposed MoT network can achieve competitive throughput and low latency, especially when the input traffic (or the on-chip parallelism) is high. Assuming $1GHz$ clock rate at $65nm$ technology, a 64-terminal MoT network can provide a 4Tbps throughput, with a total (from source to destination) switch delay of $280ps$.

2 Background and Related Work

2.1 Existing Interconnection Network Models

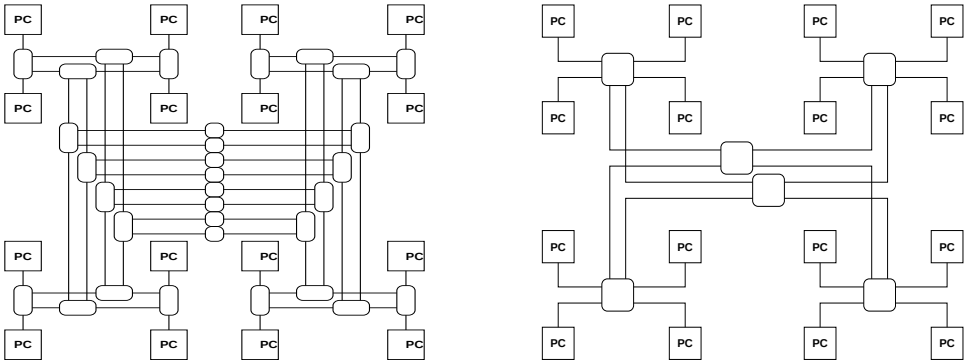
2.1.1 Hypercube

An n-dimensional hypercube, Q_n, connects $N = 2^n$ nodes by connecting a node to n other nodes. If we label the nodes from 0 to $N - 1$ in binary, a pair of nodes are connected directly if and only if their labels differ by one bit [7, 12, 17]. This connection consists of two uni-directional physical communication channels (wires). Figure 1(a) depicts the best known implementation of Q_4 in terms of area efficiency [12]. $N = 16$ nodes (PC stands for *Processing Cluster* in the figure) are connected by wires in tracks shown in the shaded areas between the PCs.



(a) 4-dimensional hypercube (b) Butterfly Logical Drawing (c) Butterfly Physical Layout

Figure 1. (a) A physical implementation of 4-dimensional hypercube. (b) Butterfly network and (c) its layout with $N = 8$ PCs as shown in [32].



(a) 2-ary 4-tree (b) Butterfly Fat Tree

Figure 2. Two types of fat trees with constant switch size. (a) k-ary n-tree with $k = 2, n = 4, N = k^n = 16$; (b) Butterfly Fat Tree with $N = 16$.

2.1.2 Butterfly

A binary butterfly network also connects $N = 2^n$ nodes as shown in Figure 1(b). The 8 PCs are connected to each other through switch nodes labeled (by their vertical layers) A, B, C, and D. For example, the connection between source 0 and destination 5 and the connection between source 6 and destination 6 are highlighted. Figure 1(c) shows the best known physical layout to implement the same butterfly network for single chip multiprocessors [32].

2.1.3 Fat Trees

Fat tree network provides multiple paths between each pair of nodes. A disadvantage of fat tree is its large switch size. The following two structures were proposed to overcome this disadvantage.

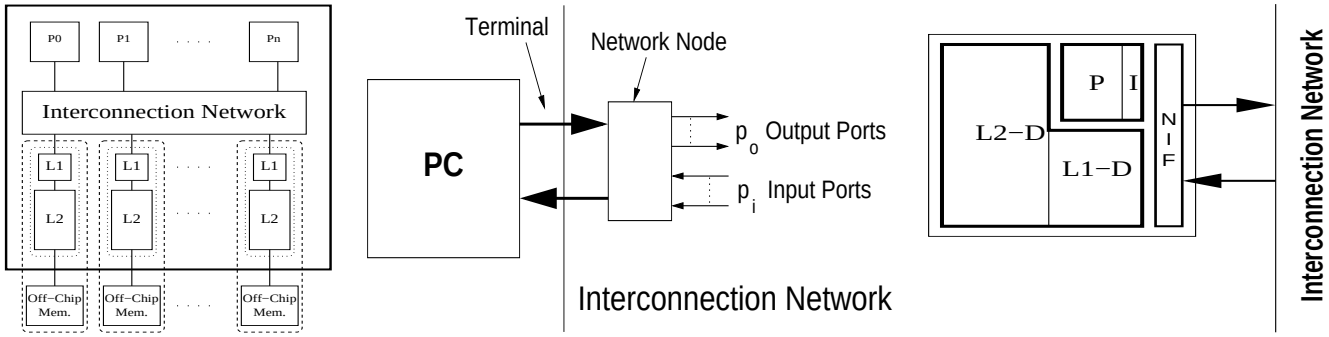
1. The *k-ary n-tree* [27] connects $N = k^n$ PCs with a fat tree of n levels as illustrated in Figure 2(a). Root nodes (small circles in the center) have k children, switch nodes (oval shape internal nodes) have k children and k parents, and there are two unidirectional links between a child and parent. Thus there are $2k$ input ports and $2k$ output ports for each switch node.

2. Figure 2(b) depicts a *butterfly fat tree* with $N = 16$ PCs. Each internal switch node (the square with no label surrounded by 4 PCs) is connected to 4 PCs and the 2 root switch nodes. Thus it has 6 input ports

and 6 output ports.

We assume that all of the above networks use *virtual channels (VCs)* in switch nodes to improve throughput. VCs act as buffers for incoming data packets that are stalled due to contention in later stages. A packet is stored in a virtual channel in the switch until an output port and physical channel toward its destination becomes available. More details on VCs and their use in interconnection networks can be found in [6, 7, 11, 27].

2.2 A Memory Architecture for Single-Chip Parallelism



(a) General Architecture (b) PC-Network Connection (c) Tentative PC Architecture

Figure 3. Memory Architecture: (a) Global memory is partitioned into modules (separated by dashed lines). Each module has its own possibly multi-level on-chip caches (within dotted lines). (b) Each PC is connected to the network by two unidirectional channels. (c) Tentatively memory modules and processors are colocated within a PC, but logically separated as shown in *(a)*. *P*: Processing elements (hardware threads), *I*: Instruction Cache, *L1-D* and *L2-D*: First and second level of data caches, *NIF*: Network interface to regulate the communication to the interconnection net work.

Using multiple memory modules (or banks) has been a common approach to increase memory bandwidth. In general, the global memory space is partitioned over the modules, and accesses to different modules are handled concurrently. The following memory structure (see Figure 3(a)) is used in the XMT single-chip parallel architecture, [23], which is designed to optimize single-task completion time. A globally shared memory space is partitioned into multiple memory modules. Each memory module consists of on-chip cache and off-chip memory portions. A universal hashing function is used to avoid pathological access patterns (similar to [10, 1, 2]). This structure completely avoids cache coherence issues because the processors do not have their private caches. However, it imposes significant challenges for the interconnection network design.

First, a hardware thread should be able to send memory requests to any memory location on the chip. Coupled with the objective of avoiding cache coherence issue, this requires placing an interconnection network between processing units and the first level of the cache and will cause a higher latency compared to cache access latencies of traditional serial processors. Multiple threads will run concurrently to hide this latency. In order to satisfy the steady and high demand of threads for data, the interconnection network needs to provide high throughput between the processing units (threads) and the memory modules.

Second, the interconnection network needs to provide low on-chip communication latency. This will allow designating fewer threads just to hide latencies and will simplify the overall design. It will also improve performance in cases where a sufficiently large number of thread contexts is not available to overlap communication with computation.

Finally, the interconnection network needs to take as little chip area as possible.

Hypercube, fat trees, and butterfly networks cannot provide the high throughput and low latency requirements for such memory structure designated for single-chip parallel processing. This is mainly due to their inability to avoid or reduce packets interference. Consider two packets going from sources s_i and s_j to destinations d_i and d_j, respectively. In a *non-interfering network*, excess traffic destined to target node d_i will not interfere with or

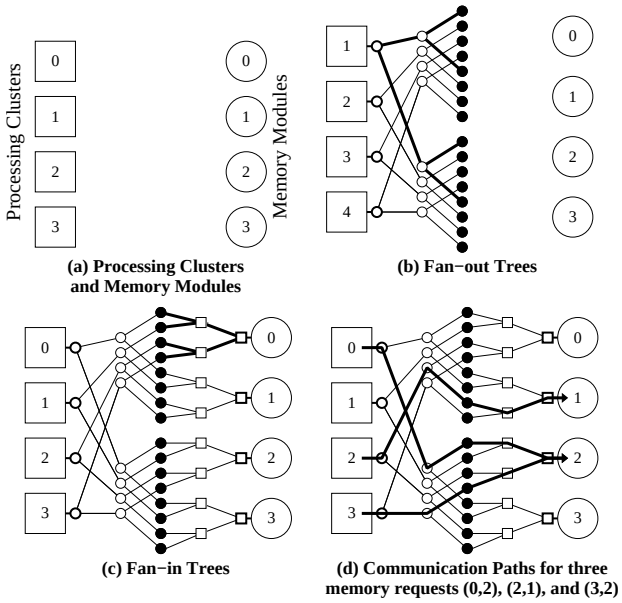


Figure 4. Mesh of Trees with 4 Clusters and 4 Memory Modules

steal bandwidth from traffic destined to target node d_j [7]. Hypercube, fat trees, and butterfly networks are not non-interfering and packet interference will cause degradation of their performance in terms of throughput and latency.

In this paper, we propose a new interconnection network implementation based on mesh of trees (MoT). The MoT structure guarantees that unless the memory access traffic is extremely unbalanced, packets between different sources and destinations will not interfere. We will demonstrate, both analytically and experimentally, that the proposed MoT network can provide high throughput with low latency.

3 Mesh of Trees Network

In earlier sections the memory architecture and its implementation challenges were overviewed. We now present the MoT network in the context of single-chip parallel processing.

3.1 Topology

The *mesh of trees* (MoT) concept is discussed in books such as [17], and papers such as [16, 9]. In an interconnection network based on their approach, functional units (processing units and memory modules) will be placed at the leaves of the trees, and a communication path involves climbing up and down some part of the tree. This approach is not interference free and would create performance bottlenecks, as children of internal nodes would compete for resources (connections in the communication path) even when their destinations are different.

We implement the MoT network in a different way as shown in Figure 4 with four processing clusters (PCs) and four memory modules (MMs). We treat the PCs as sources for packets and the MMs as destinations for convenience. Unlike the above conventional MoT approach, we put the PCs and MMs at the roots of the trees instead of the leaves. The network consists of two main structures, a set of fan-out trees and a set of fan-in trees[1]. Figure 4(b) shows the binary fan-out trees, where each PC is a root and connects to two children (we call them *up child* and *down child*), each child will have two children of their own. The 16 leaf nodes also represent the leaf nodes in the binary fan-in trees that have MMs as their roots (Figure 4(c)).

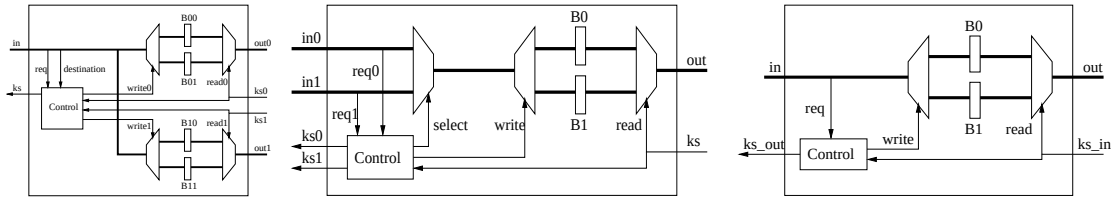
[1]They are called row and column trees in [17]. Here we use the names of fan-out and fan-in trees to convey the data flow direction.

There are two interesting properties of our MoT network. First, there is a unique path between each source and each destination. This simplifies the operation of the switching circuits and allows faster implementation which translates into improvement in throughput when pipelining a path (registers need to be added to separate pipeline stages). Second, packets between different sources and destinations will not interfere, unless the traffic is heavily unbalanced. When several packets need to reach the same destination, some packets may be queued at fan-in tree nodes. Since each node has a limited queue storage capacity, packets can be backed up to previous nodes in the fan-in trees. In extreme cases such backup can spill to the fan-out trees. This is the only case that the path to one destination can affect the path to a different destination. But for that to happen, the demand for the first destination needs to be very high, and exceed the storage capacity of the fan-in tree nodes. We will further discuss this below in the flow control section.

3.2 Routing

Routing is the process of finding a path for each packet from its source to its destination. Figure 4(d) gives the communication paths from PCs to MMs for three memory requests. Each memory request will travel from the PC (source) through a fan-out tree and then a fan-in tree before it reaches the MM (destination). There is no routing decision to be made in the fan-in trees as all packets move toward the root. In fan-out trees, routing decision is trivial from the binary representation of the destination. For example, when PC 0 sends a packet to MM 2 (10 in binary) as shown in Figure 4(d), the packet goes from the root to its down child (because of the first bit 1 in 10) and then it selects the up child (because of the 0 in the next bit position in 10) and reaches the leaf. This simple routing scheme also ensures that the fan-out tree part of the network is non-interfering. Similarly, packets with different destinations will not interfere in the fan-in trees.

3.3 Flow Control



(a) Fan-out tree primitive (b) Arbitration primitive (c) Pipeline primitive

Figure 5. Switch primitives for MoT network. Data paths are marked with thick lines. Control paths are simplified. (a) Fan-Out tree primitive: One input channel, two output channels; (b) Fan-In tree (arbitration) primitive: Two input channels, one output channel; (c) Pipeline primitive: One input channel, one output channel. Signals: *req*: Request; *ks*: Kill-and-Switch; *write*/*read*: Write and Read pointers; *B*: Storage Buffer; *select*:Result of Arbitration; *destination*: Destination address

Flow control mechanisms manage the allocation of channels and buffers to packets. Figure 5 illustrates the switching primitives in our MoT network. Each node in the fan-out and fan-in trees of the network will be implemented as the fan-out or fan-in (arbitration) primitives as shown in Figure 5 (a) and (b). The pipeline primitive in Figure 5(c) is used to divide long wires into multiple short segments.

In general, packets do not compete for resources in the fan-out trees. They stall only when the fan-in trees stall and the stall propagates to the fan-out trees. As we have explained earlier, this occurs rarely when traffic pattern is extremely unbalanced.

Arbitration in fan-in trees ensures that the two children of a node pass requests to their parent in an alternating fashion when there is a stream of incoming packets at each child. In other words, if a request from one child loses the arbitration to the other childs request in one cycle, the request is guaranteed to win the arbitration in the next cycle.

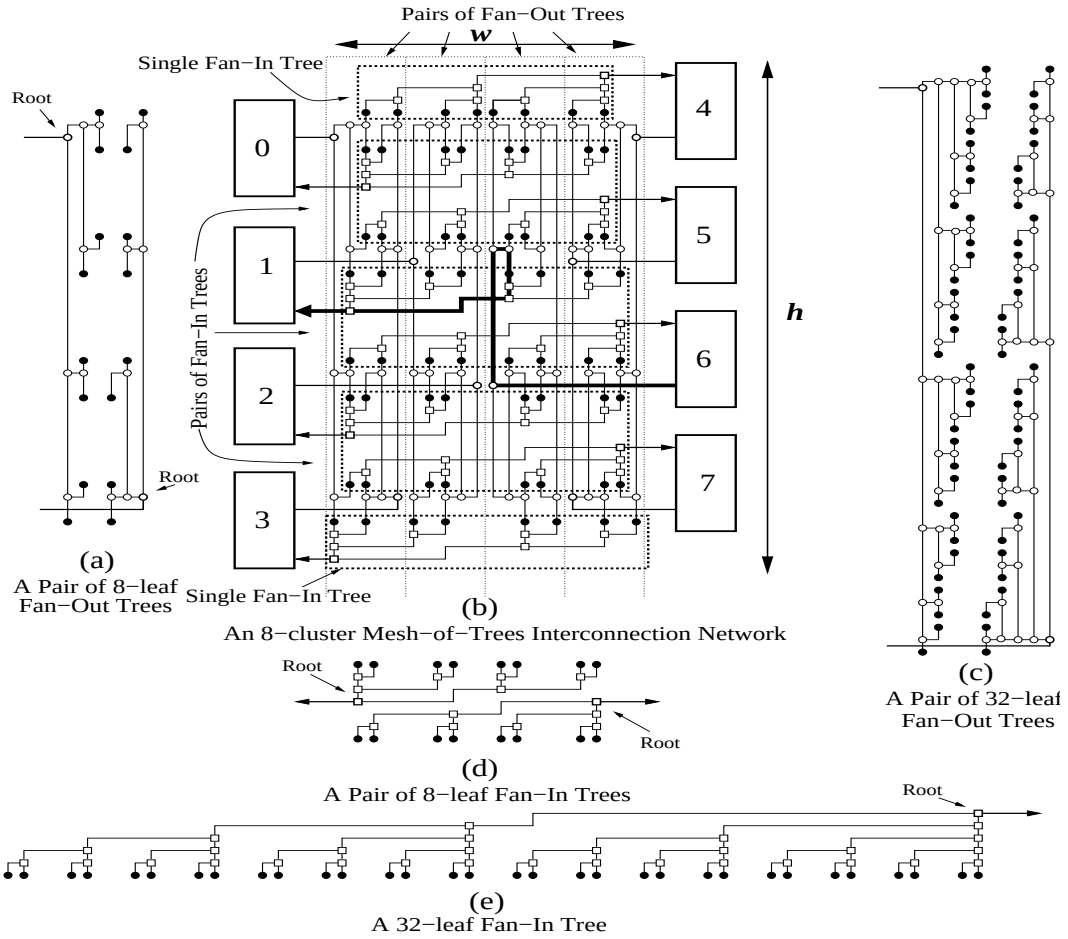


Figure 6. Detailed floorplan of the MoT interconnection network. (b): the 8-cluster MoT network floorplan; (a) and (d): details of a fan-out tree pair and a fan-in tree pair in (b); (c) and (e): layout of 32-leaf trees.

The use of local flow control mechanisms reduces the communication overhead. However, in the absence of a global *stall* signal, a node has to get the stall information from its immediate successor. A toggled *kill-and-switch* (ks) signal allows the subsequent data packet to appear at the output port. The lack of the toggle keeps the same packet at the output, effectively signaling a *stall* condition. In order to prevent data loss, our implementation uses a slightly modified version of a *relay station* [3]. Each output port has two buffers, and a small control circuit ensures that consecutive packets are stored in different buffers. If in a given cycle *ks* signal is not toggled one more data packet can be read without overwriting the stalled one.

3.4 Floorplan

Figure 6 depicts our proposed floorplan for the *MoT networks*. We first explain the layout of the fan-out and fan-in trees. Both the fan-out and fan-in trees are placed in pairs for better area utilization. Figure 6 (a) shows such a pair of 8-leaf fan-out trees for an MoT network with $N = 8$ clusters. The two root nodes of the two fan-out trees are connected to the source clusters by the thick lines. Plain empty circles are internal nodes and filled circles are leaf connections. Figure 6 (c) shows the same layout for a pair of 32-leaf fan-out trees. Figure 6 (d) shows a pair of 8-leaf fan-in tree. The filled circles are leaves. They are connected to internal nodes represented by the empty squares. Roots of the two fan-in trees are connected to the destination clusters through the connections with arrowhead. Figure 6 (e) gives the layout of one 32-leaf fan-in tree.

Figure 6 (b) shows how the fan-out and fan-in trees are placed between the eight clusters. Each pair of the fan-out trees is placed vertically and each pair of fan-in trees is laid out horizontally. At the top and the bottom, there is only one single fan-in tree each. The leaves of fan-out tree are connected to the leaves of fan-in trees. The path of a packet from cluster 6 to cluster 1 is highlighted.

4 Evaluation

We evaluate the proposed MoT network and compare it with other networks in the following four categories: Wire area, total switch delay, maximum throughput and throughput-latency relation. Table 1 describes the symbols we used in these evaluations.

Symbol	Description	Symbol	Description
A_w	Wire area	k	Network radix (for butterfly and fat tree)
b	Number of bits per channel	N	Number of terminals
d_w	Wire pitch (technology dep. parameter)	n	Network dimension
$FO4$	Technology independent *Fan-Out-4* delay	Q_n	Hypercube with n dimensions
H_{chip}	Height of the chip	$t(Q_n)$	Number of wire tracks of Q_n
$H(s,d)$	Hop count from node s to node d	v	Number of virtual channels
H_{min}	The average minimum hop count	W_{chip}	Width of the chip

Table 1. List of symbols used in the interconnection network evaluation.

4.1 Area Evaluation

The network area will be determined by the wires, and the switches and other silicon resources can be laid underneath the wires. For simplicity we use two metal layers, one for horizontal and one for vertical wires. Future technology generations will allow around ten metal layers [30]. This would reduce the area cost of all networks. Figure 7 summarizes the area comparison of different networks with parameters for 65nm technology [30]. MoT network has more wires and less wire sharing to achieve high throughput and low latency. As expected, it requires more area, for example, 60% more than the most area efficient implementation of hypercube. The following elaborates how these area values are derived.

4.1.1 Mesh of trees

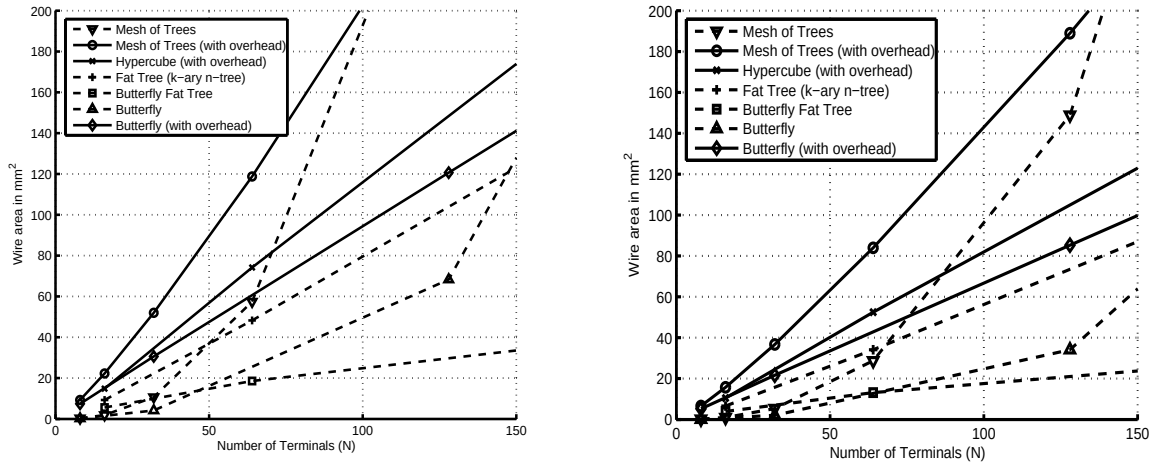
In the structure of Figure 6 (b), the width of wire area is $b{\cdot}d_w{\cdot}\frac{N}{2}{\cdot}(\log N{+}2)$ and height is $b{\cdot}d_w{\cdot}\big(\frac{N}{2}\cdot(2\log N+1)+1\big)$. Their product gives the wire area of MoT network. When we stretch the network along the vertical direction to the height of the chip in order to reach all the processing clusters, the area with such overheads becomes:

$$A_w = \frac{1}{2}\cdot b\cdot d_w\cdot H_{chip}\cdot N\cdot(\log N+2)$$

4.1.2 Hypercube

As shown in Figure 1(a), the chip area is $N\cdot(s+w_w)^2$, where s is the size of a PC, $w_w = 2\cdot t(Q_{n/2})\cdot b\cdot d_w$ is the width of the wire area between two PCs. The constant 2 is due to the use of unidirectional channels, $t(Q_{n/2})$ is the number of tracks in such area, and $n=\log N$. We use the formula for $t(Q_{n/2})$ from [12], subtract the area of PCs, and obtain hypercube network's wire area:

$$A_w = 4\cdot b\cdot d_w\cdot W_{chip}\cdot\sqrt{N}\cdot\left\lfloor(2/3)\sqrt{N}\right\rfloor - 4\cdot\left(\left\lfloor(2/3)\sqrt{N}\right\rfloor\cdot b\cdot d_w\right)^2$$



(a) Wire Area with 65nm Global Wires

(b) Wire Area with 45nm Global Wires

Figure 7. Wire Area Comparison at (a) 65nm Technology and (b) 45nm Technology. $H_{chip} = W_{chip} = 20mm$**,** $b = 80$**, and** $d_w = 290nm$ **for 65nm Technology and,** $210nm$ **for 45nm Technology [30].**

4.1.3 Butterfly

The number of wire tracks required in both dimensions can be obtained from the layout of [32]. Similar to the MoT approach, their product gives the wire area and when we include the overheads, the area can be expressed as:

$$A_w = b \cdot d_w \cdot H_{chip} \cdot (2N + \log N - 3)$$

4.1.4 Fat trees

The wire area does not change for k-ary n-trees with different values of k and n as long as $N = k^n$ is kept constant. We calculate the total wire area by iteratively adding wires starting from the root. Without overheads, the wire area is:

$$A_w = b \cdot d_w \cdot (N \cdot W_{chip}(1 - 2^{-\lfloor \frac{\log N}{2} \rfloor}) + N \cdot H_{chip}(1 - 2^{-\lfloor \frac{\log N - 1}{2} \rfloor}))$$

The area of butterfly fat tree can be obtained similarly:

$$A_w = b \cdot d_w \cdot \sqrt{N} \cdot \left(W_{chip} \cdot \left\lfloor \frac{\log N}{2} \right\rfloor + H_{chip} \cdot \left\lfloor \frac{\log N - 1}{2} \right\rfloor \right)$$

4.2 Register Count

Routing switches consist of several data registers of b-bits each, and some control circuit that handles resource allocation, and forward and backward signaling. In typical routing switches [7], the control circuit handles both routing (from one input to which output) and arbitration (from which input to one output) at the same time. In these switches, the effective number of inputs and outputs are multiplied by the use of *virtual channels* for increased performance. Each such such virtual channel uses one b-bit register for one data packet.

In our proposed MoT network, routing and arbitration are are handled by different primitives. Furthermore, each primitive has either one or two input and output ports and no virtual channels, making the decision logic simpler. Therefore, we assume that the control circuitry of our primitive circuits consume negligible area compared to b-bit data registers, especially for high values of b, such as 80 used in Figure 7.

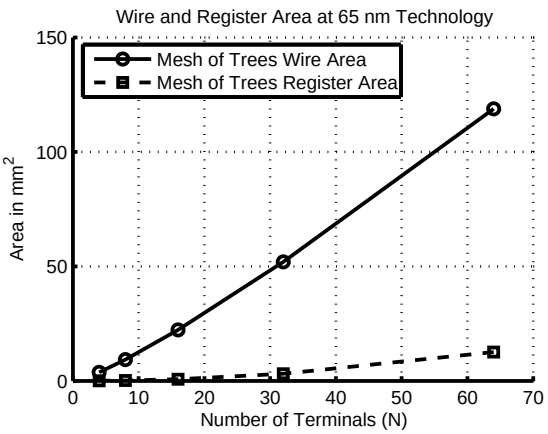


Figure 8. Wire and Register Area for Mesh of Trees at 65nm Technology

Figure 8 depicts the area of wires and data registers for different N values for $65nm$ technology. It shows that the assumptions of wires dominating total area is valid. Table 2 compares the register count for MoT and other networks as a measure of cost. MoT requires more registers when other networks has fewer VCs. However, when we scale the value of v to improve throughput, other networks need more registers shile still not being able to provide the same throughput as MoT (see Table 3). The following shows how these numbers are derived.

N	4	8	16	32	64
MoT	72	336	1440	5952	24192
Butterfly $v = 4$	128	384	1024	2560	6144
Butterfly Fat Tree $v = 4$	96	192	384	768	1536
Hypercube $v = 4$	64	192	512	1280	3072
Fat Tree $(k = 4)$ $v = 4$	160	304	576	1104	2144
Butterfly $v = N$	128	768	4096	20480	98304
Butterfly Fat Tree $v = N$	96	384	1536	6144	24576
Hypercube $v = N$	64	384	2048	10240	49152
Fat Tree $(k = 4)$ $v = N$	160	608	2304	8832	34304

Table 2. Register count for MoT and other networks

4.2.1 Mesh of Trees

As shown in Figures 4 and 6, the network consists of N fan-out and N fan-in trees, each with $(N - 1)$ nodes. The leaves do not contain switching circuits, since they are only wire connections. Based on the primitive circuits shown in Figure 5, the total number of b-bit registers is equal to 6 $N(N-1)$. The number of transistors per one-bit register varies between 8 and 12 based on their design [29]. Assuming 10 transistors per bit, the total number of transistors of data registers in MoT network is $60 \cdot b \cdot N(N - 1)$. Transistor density estimation for a particular technology node is from [30].

4.2.2 Hypercube

Hypercube has N switching nodes, each with $\log N$ input and output ports. Each of the $2 \cdot N \log N$ input and output ports contains v data registers [7], one per virtual channel. Therefore, the total number of b-bit registers is equal to $2 \cdot v \cdot N \log N$.

4.2.3 Butterfly

Similarly, the butterfly network has a total of $4 \cdot N \log N$ input and output ports with v virtual channels each, and therefore $4 \cdot v \cdot N \log N$ b-bit registers.

4.2.4 Fat Trees

The k-ary n-tree with $N = k^n$ terminals has N root nodes with $2 \cdot k \cdot N$ total ports, and $N \log_k N$ internal nodes with $4 \cdot k \cdot N \log_k N$ total ports. In total they require $2 \cdot v \cdot k \cdot (2 \cdot N + \log_k N)$ b-bit registers.

In the butterfly-fat-tree, the total number of switches approaches to $N/2$ as N grows [8, 24, 11]. Each switch node has 6 input and 6 output ports. Therefore the total number of b-bit registers will be approximately $6 \cdot v \cdot N$.

4.3 Total Switch Delay

The total switch delay is measured as the product of the *average number of hops* from source to destination and the switch delay of the switch circuit. Switch delay for MoT is derived from the synthesis results using Synopsys tools and others are computed following the formulas given in the literature [26, 11]. This switch delay is measured in terms of $FO4$, a technology independent delay unit that represents the delay of an inverter driving four identical inverters. As Figure 9 shows, a 64-terminal configuration of the proposed MoT network has the smallest total switch delay, 2.8 to 6.0 times less than others. The following describes how we obtain the switch delay and average number of hops.

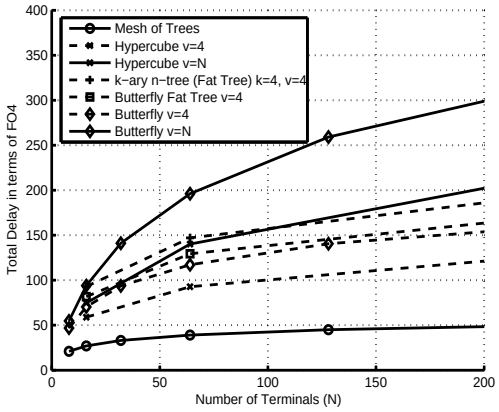


Figure 9. Technology independent total switch delay comparison.

4.3.1 Mesh of Trees

We synthesize the switch primitives described in Figure 5 in Synopsys. The timing analysis shows that the circuit delays vary between 2.25 $FO4$ and 2.81 $FO4$. For simplicity we assume a conservative constant delay of 3 FO4. From the MoT topology, it is easy to see that the number of hops is $2 \log N + 1$ for all source and destination pairs.

4.3.2 Hypercube

The switch node of a hypercube that connects N processing clusters has $\log N + 1$ input and output ports. We assume a typical setting of $v = 4$ virtual channels, and an aggressive one $v = N$ to improve throughput, and compute the switch delays based on [26]. Under uniform traffic pattern, the average number of hops for hypercube is $\frac{1}{2} \log N$ [7].

4.3.3 Butterfly

In the binary butterfly networks we described in Section 2.1, each internal switch node has two input and two output ports. The switch delay is computed similar to the hypercube [26]. The number of hops is always equal to $\log N + 1$.

4.3.4 Fat trees

Internal switches in a k-ary n-tree have $2k$ input and $2k$ output ports. The root switches have k input and k output ports. Switch delays are computed based on [26] with $v = 4$ and $v = N$ virtual channels. Similarly, we calculate the switch delay for butterfly fat tree based on the analysis in [11].

Average hop count for the k-ary n-tree is listed in the following equation, when we set $k = 4$, the equation gives the average hop count for the butterfly fat tree.

$$H_{min} = \frac{1}{N^2} \sum_{s=0}^{N-1} \sum_{d=0}^{N-1} H(s,d) = \frac{2 \cdot (k-1)}{N \cdot k} \cdot \sum_{t=0}^{\log_k N} t \cdot k^t$$

4.4 Maximum Throughput

In order to evaluate the maximum throughput provided by each network model, we assume the maximum packet generation rate of one packet per cycle (1.0 *ppc*) at each input port of the network[2]. At this generation rate, the network will saturate with packets, and the injection and delivery rates will come to balance at the maximum throughput. We assume uniform traffic pattern, which is expected for the memory architecture described in Section 2.2. Uniform traffic pattern is a reasonable assumption due to the use of hashing mechanism, which has an effect similar to randomization that distributes the memory accesses evenly among modules [21, 10, 1, 2].

We obtain the results for hypercube, butterfly, and MoT networks from theoretical analysis and simulation. The results for fat tree networks are from [27] and [25]. As one can see from Table 3, the proposed MoT network can provide the highest maximum throughput, which is 76% and 28% higher than butterfly and hypercube with $v = 4$ virtual channels, and 3% and 16% higher than butterfly and hypercube with 64 virtual channels, respectively.

4.4.1 Mesh of Trees

Each PC injects 1.0 *ppc* into its fan-out tree (see Figure 4 (b)). Each leaf will receive $1/N$ *ppc* for the uniformly injected traffic. In the fan-in tree, the $1/N$ *ppc* traffic will accumulate to 1.0 *ppc* at the root node. Therefore, MoT network is capable of delivering the maximum 1.0 *ppc* at each destination port.

In simulation, due to temporary imbalances in traffic caused by the stochastic nature of the generation process, 1.0 *ppc* is not expected. As we increase N, the number of terminals, there will be more pipeline stages between the leaves of fan-out tree and fan-in tree. This will relieve some of this imbalance and provide higher throughput. In fact, we observe the maximum throughput of 0.951 *ppc* per port for $N = 16$ goes up to 0.977 *ppc* when $N = 64$.

4.4.2 Hypercube

Since a single port connects the PCs to the network, and the highest injection rate through that port is equal to 1.0 *ppc*, the maximum achievable throughput is equal to 1.0 *ppc*. In hypercube network, data packets from different sources to different destinations may compete for some common channels and cause interference and throughput decrease. When the number of virtual channel is fixed (e.g., at a typical value $v = 4$), the maximum throughput decreases as N increases because there might be more interference. Adding more virtual channels will improve throughput at the cost of increased switch delay. The simulation results shown in Table 3 support these analysis.

[2]Note that in several other studies of interconnection networks, long data packets are divided in shorter units, called *flits*. Since our interconnection network is in a critical path between the processors and the cache memory, we do not cut a packet in shorter flits and assume that the entire packet of b bits is injected at once [1]

Configuration	Max Throughput	Configuration	Max Throughput
Hypercube $N = 16$ $v = 4$	0.777	Butterfly $N = 16$ $v = 4$	0.602
Hypercube $N = 64$ $v = 4$	0.763	Butterfly $N = 64$ $v = 4$	0.553
Hypercube $N = 16$ $v = 16$	0.787	Butterfly $N = 16$ $v = 16$	0.861
Hypercube $N = 64$ $v = 64$	0.843	Butterfly $N = 64$ $v = 64$	0.946
Fat Tree $N = 256$ $k = 4$ $n = 4$ $v = 2$	0.55	BFT $N = 64$ $v = 4$	0.28
Fat Tree $N = 256$ $k = 4$ $n = 4$ $v = 4$	0.72	BFT $N = 64$ $v = 8$	0.30
Mesh of Trees (theoretical)	1.0	Mesh of Trees $N = 16$	0.951
Mesh of Trees $N = 32$	0.963	Mesh of Trees $N = 64$	0.977

Table 3. Maximum throughput (in *packets per cycle per port*) provided by different networks (N: numbner of terminals, v: number of virtual channels, BFT: *Butterfly Fat Tree*.).

4.4.3 Butterfly

Under uniform traffic, the theoretical maximum throughput of the butterfly network is 1.0 *ppc* per port. However, similar to hypercube, the actual throughput might be reduced due to packet interference. Table 3 reports the maximum throughput for butterfly network with different settings. Interestingly, it outperforms hypercube with the presence of large number of virtual channels.

4.4.4 Fat trees

Because of its sufficiently many physical channels, the k-ary n-tree network can provide 1.0 *ppc* per port when an off-line routing algorithm is used to avoid packet interference. However, on-line routing algorithms cannot guarantee interference-free packet transmission and this results in throughput less than 1.0 *ppc*. Butterfly fat tree has fewer physical channels than the k-ary n-tree and therefore will have lower throughput. Results from [25] and [27] are summarized in Table 3.

4.5 Throughput and Latency

As traffic in the network increases, packets will experience longer latencies. We follow the guidelines in [7] to design simulations in order to evaluate the throughput and latency of various network models under different input traffic. We are particularly interested in the case when the input traffic, or the on-chip parallelism, is high. Hypercube and butterfly networks are simulated on the simulator provided by [7] with $N = 64$ terminals, and different number of virtual channels, namely a typical $v = 4$ setting and an aggressive $v = 64$ setting. Router switches have three cycle switch latency per *speculative virtual channel router* design of [26]. MoT network is simulated using an RTL SystemC simulator we implemented.

First, to validate our own simulator, we implement a single-cycle butterfly switch similar to the switch primitives shown in Figure 5. We then use our simulator to simulate the butterfly network with $N = 64$ and this switch. The results are reported in Figure 10 as the two curves marked *systemC BF*. The simulation results from [7] on the butterfly network with the $N = 64$ $v = 2$ configuration and single-cycle switch latency are also reported in Figure 10 as the curves marked *booksim BF*. One can see that these curves match reasonably well which indicates the accuracy of our simulator.

We vary the input traffic from the low 0.1 *ppc* per port to the maximum 1.0 *ppc*. Traffic is generated as a Bernoulli process and the packet destinations are uniformly random. The latency of a packet is measured as the time from it is generated to the time it is received at the destination, which includes the waiting time at the source queue. For each input traffic rate, we use different seeds to generate a set of traffic with the same traffic rate. These input traffic sets are injected to simulators for each network model and the average throughput and latency are reported in Figure 10.

MoT network provides competitive throughput and latency and has a clear advantage over others when the input traffic is high. More importantly, MoT network has a more predicable latency when the input traffic varies. For example, when we increase the input traffic from 0.1 *ppc* per port to 0.9 *ppc*, the hypercube latency increases

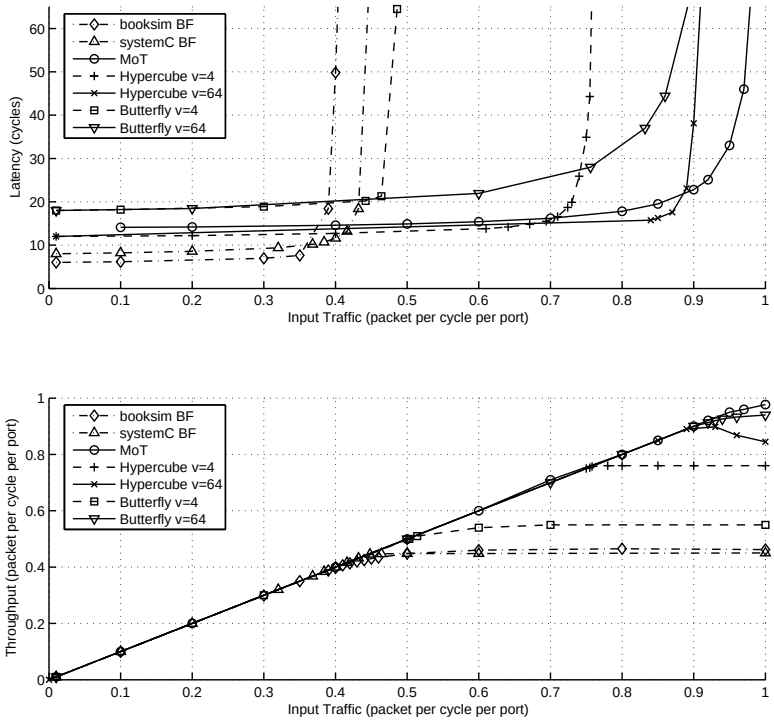


Figure 10. Throughput and latency of various networks for $N = 64$ terminals.

by a factor of 3.2,butterfly network latency increases by a factor of 3.9,while MoT latency increases only by a factor of 1.6.

4.6 Delay Effects of Long Wires

In the existence of long wires at current and future technologies, wire delay becomes more and more important because it results in an increase in clock period and thus reduces throughput in terms of packets per second. One method to deal with such adverse effects of long wires is to cut them into shorter segments by inserting pipeline stages ([4, 13, 20]). This will maintain the throughput at the cost of longer latency in terms of clock cycles.

We have modified our network simulator to account for the additional cycles that were created by pipelining. The results show that for MoT network, as expected, the throughput in terms of *ppc* does not change and the latency increases by the same amount for all different rates of traffic load as shown in Figure 11. Now we report the results on the long wire effect on latency for MoT, butterfly and hypercube networks.

Given a hypothetical clock period of $16FO4$ as per [14, 28], for example), ITRS estimations [30] imply that this corresponds to a clock frequency of approximately 7.2 GHz at $65nm$ technology. If we neglect the circuit switch delay, the *maximum traversable wire distance per cycle* $L_{crit} = 1.22\ mm$. The switch delay of MoT primitives is less than $3FO4$ (Section 4.3), and the remaining $13FO4$ corresponds to 1.1 mm. Considering such switch delay will not impact our results on the comparison of different networks. Finally, we assume that global wires are used, wire delay is quadratically proportional to the wire length, and $H_{chip} = W_{chip} = 20\ mm$.

Based on the layouts in Figures 6, 1(a), and 1(c), we measure the distance between each network switch circuit. For MoT we use the primitive circuits (Section 3.3), and for butterfly and hypercube networks we use the pipelined switches described in [26] and a one-cycle pipeline stage to cut the wires. Based on the L_{crit} value we determine the number of pipeline stages that we need to insert. The minimum latency between a source and destination is computed by adding the cycles spent in a switch node and pipeline stages on wires.

Table 4 reports the *average minimum latency* for the three network models with different number of terminals. The average is taken over all the possible source-destination pairs[3]. MoT outperforms both hypercube and butterfly. For example, then there are 32 terminals, MoT network has average latency 11% and 23% lower than a hypercube and butterfly respectively.

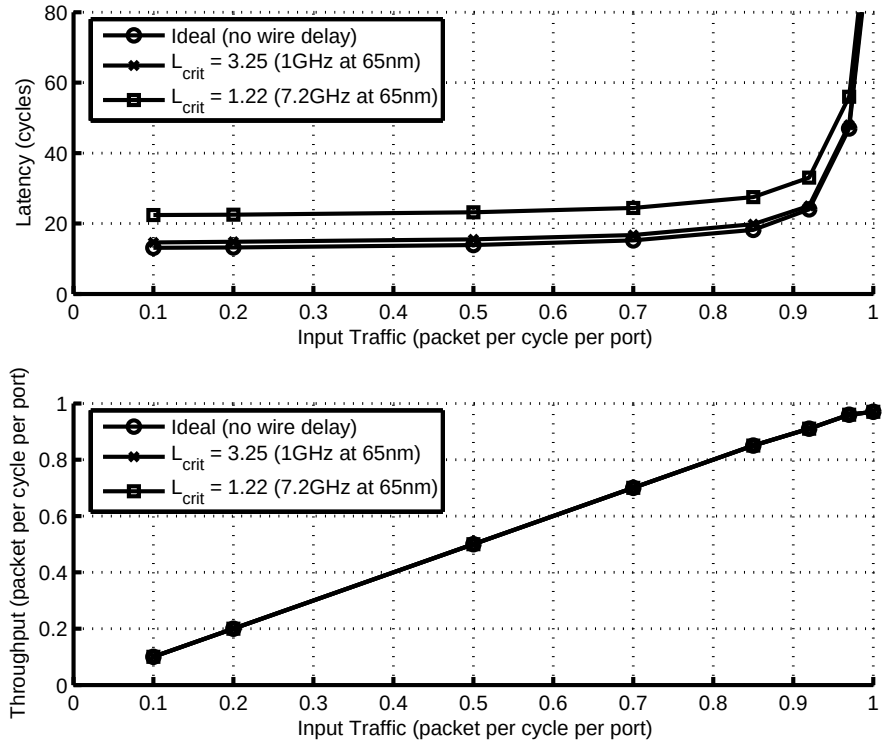


Figure 11. Latency and Throughput results of MoT network for various values of L_{crit} with $N = 64$ terminals.

N	8	16	32	64	128
MoT	13.0	16.0	19.1	22.3	27.6
Hypercube	15.5	19.0	21.5	24.0	26.0
Butterfly	17.0	N/A	25.0	N/A	34.5

Table 4. Average minimum packet latency (in clock cycles). 65nm Technology, $H_{chip} = W_{chip} = 20\ mm$, $L_{crit} = 1.22\ mm$

5 Conclusion and Future Work

A high-performance interconnection network between processors and memory modules is desirable for parallel computing. First, it enables support of easy parallel programming approaches, e.g., PRAM-like [31, 22] that cannot be supported otherwise [5]. Second, it allows feeding data to the functional units at a sufficient rate. The

[3]For even powers of 2, the butterfly network in 1(c) needs to be modified, therefore we omitted the results for those values.

necessity of having a high data rate is amplified by the architectural choice of XMT to put the network between the processing units and the first level of data cache.

We introduce a certain *Mesh of Trees* (MoT) interconnection network and compare it with traditional networks in terms of wire area, total switch delay, maximum throughput and latency-throughput relation. From a broader perspective, our most interesting results are probably the ones related to the effective throughput. A recurring problem in networking and communication is that of achieving good sustained throughput in contrast to just high theoretical peak performance that does not materialize for typical workload. This is where our quantitative results demonstrate a clear advantage for the proposed MoT implementation. Particularly, at $65nm$ technology, a 64-terminal MoT network operating at $1GHz$ clock rate, provides up to $62G$ words ($4T$ bits) per second throughput. Between the source and destination terminals, the total switch delay is approximately $280ps$. Ongoing studies on wire delays will allow more accurate latency and throughput analysis.

The interconnection network of multi-chip Tera parallel computer has a bisection bandwidth of one 64-bit word per cycle per processor [1]. By providing the same per-cycle bandwidth and a matching actual throughput within the chip, MoT eliminates the clock cycle limitations of inter-chip communication. This results in much faster clock rates and significantly lower memory access latency.

It would be useful to better understand the cost-performance trade-offs of interconnection network designs for single-chip parallel processors with a large number of processing units. According to our results, the proposed MoT network outperforms all of the existing approaches in terms of latency and throughput, while this comes at a price of increased area. We plan to study opportunities for reducing the wire area, and the effects of wire delays on packet latency, as well as other cost factors such as the area cost of circuit resources and power requirements. Meanwhile we will also study different approaches, e.g. two parallel butterfly networks, that might provide competitive performance at a lower area cost. The study and results presented in this paper lay a foundation for such future work.

Finally, the use of MoT network is not limited to parallel processors. As the number of functional units grow in system-on-chip applications, on-chip networks will be required to satisfy the communication needs. MoT network may provide an alternative solution, where high-throughput and low-latency communication is needed.

References

[1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The tera computer system. In *Proc. Int. Conf. On Supercomputing*, pages 1–6, 1990.

[2] P. Bach, M. Braun, A. Formella, et al. Building the 4 processor SB-PRAM prototype. In *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, volume 5, pages 14–23, Jan. 1997.

[3] L. P. Carloni, K. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 301 – 315, 1999.

[4] P. Cocchini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 268–273, 2002.

[5] D. E. Culler and J. P. Singh. *Parallel Computer Architecture*. Morgan Kaufmann, 1999.

[6] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3(2):194–205, Mar. 1992.

[7] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Fransisco, CA, 2004.

[8] A. DeHon. Compact, multilayer layout for butterfly fat-tree. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 206–215, 2000.

[9] A. DeHon and R. Rubin. Design of FPGA interconnect for multilevel metallization. *IEEE Trans. VLSI Syst.*, 12(10):1038–1050, 2004.

[10] A. Gottlieb, R. Grishman, C. Kruskal, K. McAuliffe, L. Rudolph, and M. Snir. The NYU ultracomputer–designing an MIMD shared memory parallel computer. *IEEE Trans. Comput.*, pages 175–189, Feb. 1983.

[11] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh. Structured interconnect architecture: A solution for the non-scalability of bus-based SoCs. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 192 – 195, 2004.

[12] R. I. Greenberg and L. Guan. On the area of hypercube layouts. *Information Processing Letters*, 84:41–46, 2002.

[13] S. Hassoun, C. J. Alpert, and M. Thiagarajan. Optimal buffered routing path constructions for single and multiple clock domain systems. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 247 – 253, 2002.

[14] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proc. IEEE*, 89(4):490 – 504, Apr. 2001.

[15] D. J. Kuck. A survey of parallel machine organization and programming. *Computing Surveys*, pages 29–59, 1977.

[16] F. T. Leighton. New lower bound techniques for VLSI. In *Proc. Of the 22nd IEEE Symposium on Foundations of Computer Science*, pages 1–12, 1981.

[17] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

[18] C. E. Leiserson. Fat trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, Oct. 1985.

[19] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, et al. The network architecture of the connection machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.

[20] R. Lu, G. Zhong, C. Koh, and K. Chao. Flip-flop and repeater insertion for early interconnect planning. In *DATE '02: Proceedings of the Conference on Design, Automation and Test in Europe*, page 690, 2002.

[21] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21:339–374, 1984.

[22] D. Naishlos, J. Nuzman, C.-W. Tseng, and U. Vishkin. Towards a first vertical prototyping of an extremely fine-grained parallel programming approach. *Theory of Computer Systems*, 2003. Special Issue of SPAA 2001.

[23] J. Nuzman. Memory subsystem design for explicit multithreading architectures. Master's thesis, University of Maryland, 2003.

[24] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages V–217 – V–220 vol.5, 2003.

[25] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Evaluation of MP-SoC interconnect architectures: A case study. In *IEEE International Workshop on System-On-Chip for Real-Time Applications*, pages 253 – 356, 2004.

[26] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 255–266, 2001.

[27] F. Petrini and M. Vanneschi. Performance analysis of wormhole routed k-ary n-trees. *International Journal of Foundations of Computer Science*, 9(2):157–178, 1998.

[28] T. M. Pinkston and J. Shin. Trends toward on-chip networked microsystems. Technical Report CENG-2004-17, University of Southern California, 2004.

[29] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits, A Design Perspective*. Prentice Hall, second edition, 2003.

[30] Semiconductor Industry Association. The international technology roadmap for semiconductors. http://public.itrs.net/Files/2003ITRS/Home2003.htm, 2003.

[31] U. Vishkin, S. Dascal, E. Berkovich, and J. Nuzman. Explicit multi threading (XMT) bridging models for instruction parallelism. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 140–151. ACM, 1998.

[32] T. T. Ye and G. D. Micheli. Physical planning for on-chip multiprocessor networks and switch fabrics. In *Proceedings of the Application-Specific Systems, Architectures and Processors (ASAP)*, pages 97 – 107, 2003.

[33] C.-H. Yeh. Optimal layout for butterfly networks in multilayer VLSI. In *International Conference on Parallel Processing (ICPP)*, pages 379 – 388, 2003.