International Journal of Foundations of Computer Science

PERFORMANCE ANALYSIS OF WORMHOLE ROUTED K-ARY N-TREES

FABRIZIO PETRINI and MARCO VANNESCHI*
Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy,
tel +39 50 887228, fax +39 50 887226
e-mail: {petrini,vannesch}@di.unipi.it

Received 27 October 1995
Revised 14 August 1997
Communicated by D. F. Hsu, M. Palis and D. S. L. Wei

ABSTRACT

The past few years have seen a rise in popularity of massively parallel architectures that use fat-trees as their interconnection networks. In this paper we formalize a parametric family of fat-trees, the k-ary n-trees, built with constant arity switches interconnected in a regular topology. A simple adaptive routing algorithm for k-ary n-trees sends each message to one of the nearest common ancestors of both source and destination, choosing the less loaded physical channels, and then reaches the destination following the unique available path. Through simulation on a 4-ary 4-tree with 256 nodes, we analyze some variants of the adaptive algorithm that utilize wormhole routing with 1, 2 and 4 virtual channels. The experimental results show that the uniform, bit reversal and transpose traffic patterns are very sensitive to the flow control strategy. In all these cases, the saturation points are between $35-40\%$ of the network capacity with 1 virtual channel, $55-60\%$ with 2 virtual channels and around 75% with 4 virtual channels. The complement traffic, a representative of the class of the congestion-free communication patterns, reaches an optimal performance, with a saturation point at 97% of the capacity for all flow control strategies. In this case virtual channels are of little help and the average network latency experiences an increase proportional to the number of virtual channels, due to the multiplexing of several packets onto the same physical links.

Keywords: Interconnection networks, fat-trees, k-ary n-trees, routing, virtual channels, flow control, congestion-free patterns.

1. Introduction

The fat-tree is an indirect interconnection network based on a complete binary tree. Unlike traditional trees in computer science, fat-trees resemble real trees, because they get thicker near the root. A set of processors is located at the leaves of the fat-tree and each edge of the underlying tree corresponds to a bidirectional

*A preliminary version of this paper appeared in the Proceedings of the 11th International Parallel Processing Symposium, IPPS'97 channel between parent and child. A channel consists of a bundle of wires and the number of wires in a channel is called its capacity. These capacities are determined by how much hardware we can afford: this means that a fat-tree is parameterized not only in the number of processors, but also in the communication bandwidth it can support. An example of fat-tree is shown in Figure 1. The processors at the leaves are connected to the internal switches and the channel capacity doubles at each level.

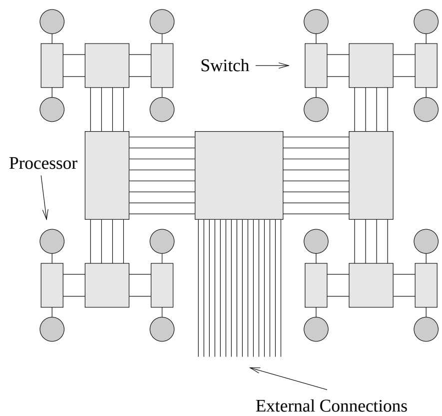

Fig. 1. The structure of a fat-tree. Processors are located at the leaves, while internal nodes contain switches. The channel capacity doubles at each level. At the root there are some external connections available to recursively build a bigger network or to interface it to the external world.

Routing on a fat-tree is relatively easy, since there is a unique minimal path between a pair of processors $\langle i, j \rangle$: a message going from i to j goes up the internal switches of the tree until it finds the nearest common ancestor and then down to j. In the presence of channels with multiple capacity, the flow control algorithm can pick one of these wires in order to evenly distribute the messages and to minimize congestion.

Fat-trees have many nice theoretical properties. Leiserson[24] proved that, for an arbitrary message set M, *off-line* scheduling[a] can be done optimally within a logarithmic factor of the number of processors. That is, M can be scheduled in a group of delivery cycles $M_1, M_2, \dots M_d$ such that $d = O(\lambda \log N)$, where N is the number of processors and λ is the *load factor*[b].

Also, fat-trees are hardware-efficient networks. The *universality theorem* from Leiserson states that a universal fat-tree[c] of a given volume can simulate any other interconnection network of equal volume with only a polylogarithmic factor increase in the time required. This theorem is proved using a three-dimensional VLSI

[a] In the off-line scheduling problem the communication pattern is known in advance.

[b] The load factor of a set of messages is the largest ratio over all channels in the fat-tree of the number of messages that must pass through the channel divided by the capacity of the channel. The load factor of a set of messages is thus a lower bound on the delivery time.

[c] The capacities of the channels of a universal fat-tree grow exponentially as we go up the tree from the leaves, doubling from one level to the next, but slow down near the root at a rate of $\sqrt[3]{4}$.

model that incorporates wiring as direct cost: "wire" and "pin" limitations are modeled directly by the communication limits imposed by the surface area of a three-dimensional region.

These results have been extended to the more interesting *on-line*[d] case[16]: if a set of messages has load factor λ on a fat-tree with N processors, the number of delivery cycles required is $O(\lambda + \log N \log\log N)$ with probability $1 - O(1/N)$. The proof is based on a randomized routing algorithm. Unlike Valiant's classic scheme[38] for hypercubes, that sends each message to a randomly chosen destination and, from there, to its true destination, the algorithm repeatedly attempts to deliver a randomly chosen set of the messages.

The arity of the internal switches of the fat-tree increases as we go closer to the root: this makes the physical implementation of these switches unfeasible. For this reason some alternative constructions have been proposed that use building blocks with fixed arity.[25] These solutions trade connectivity with simplicity: incoming messages at a given switch in a "full" fat-tree may have more choices in the routing decision than in a corresponding network with fixed-arity switches. DeHon[11] provided an in-depth study of implementation problems, as wiring and packaging complexity and fault tolerance.

Orthogonal fat-trees are an alternative formulation that uses constant size elements. The elegant recursive construction developed by Valerio *et al.*[37] tries to maximize the number of processors when the degree of the internal switches and the diameter of the network are physically constrained. The basic entity of this network is a two level fat-tree, that is obtained from complete sets of mutually orthogonal Latin Squares. Furthermore, the internal nodes of the orthogonal fat-tree can be replaced with rings.[36]

Fat-trees have been adopted by many research prototypes and commercial machines. The data network of the Connection Machine CM-5 uses two distinct fat-trees.[26] The network is composed of routing chips that have either 2 or 4 parent connections. The hierarchical nature of the fat-tree is exploited to partition the CM-5 in dedicated subnetworks whose communication traffics do not interfere between them.

The Data Diffusion Machine (DDM) is a virtual shared memory architecture that implements a hierarchical COMA cache coherence protocol.[31] The distributed directory, which typically becomes a bottleneck in hierarchical bus-based implementations, is implemented in the internal switches of a fat-tree.[32]

The communication chip Elite is the basic building block of the Meiko CS-2 network.[30] This network takes the form of a quaternary fat-tree. Its design is based on a multistage network and has the property that the overall communication bandwidth remains constant at each level. The CS-2 uses a randomized routing algorithm with header striping. The communication processor Elan, which interfaces each processing node to the network, attaches at the beginning of each outgoing message a string that is used by the communication processors to route the message

[d] In the on-line case messages are spontaneously generated by processors and routed "on the fly" by the communication switches.

in the ascending and descending phases. Other references to fat-trees include.[18] [20]

Unfortunately, not much is known on the communication performance of the fat-trees. Most of the literature deals with the CM-5 and focuses on raw network performance.[22,27,29] Typical communication patterns include simple sends and ping-pong between pairs of nodes. Block permutations of data and grid shifts have been shown to have little or no contention on the CM-5. This makes the data network very efficient for regular communication patterns commonly used in numerical algorithms as the FFT. Heller[19] provided an analytical description of a class of such permutations, defined as *congestion-free*. These communication patterns can be used to solve important problems, including all-to-all personalized broadcast and the normal hypercube algorithms.

Modern parallel routers significantly reduce average latency by using wormhole routing.[6] Wormhole routing refers to a flow control strategy that divides each packet in elementary units called *flits* and advances each flit as soon as it arrives at a node, in a pipeline fashion. Wormhole routing is attractive because it reduces the latency of message delivery compared to store and forward routing and requires only a few flit buffers per node. Network throughput of wormhole routing can be increased by organizing the flit buffers associated with each physical channel into several virtual channels.[7] These virtual channels are allocated independently to different packets and compete with each other for the physical bandwidth. This decoupling allows active messages to pass blocked messages using network bandwidth that would otherwise be wasted.

The remainder of this paper is organized as follows. Section 2 presents a recursive definition of fat-trees, and focuses on a parametric family of networks, the k-ary n-trees. k-ary n-trees are symmetric networks that borrow from the k-ary n-butterflies the topology of the internal switches. Their main topological properties are utilized to define a randomized and an adaptive routing algorithm. Section 3 presents a detailed simulation model for studying the impact of wormhole routing and virtual channels on the adaptive algorithm. The communication patterns adopted as benchmarks in the experimental evaluation are described in section 4. Using the simulation model and the benchmarks, in section 5 we evaluate the communication performance of a 4-ary 4-tree with 256 nodes. The experiments compare flow control strategies with 1, 2 and 4 virtual channels. An overview of the experimental results is provided in section 6 and some concluding remarks are given in section 7.

2. k-ary n-trees

In this section we formalize the fat-trees, giving a recursive definition that is general enough to embed many different topologies that are often quoted as fat-trees. We then turn our attention to a particular subclass: the k-ary n-trees. As the k-ary n-cubes and the k-ary n-butterflies,[8] the k-ary n-trees are a parametric family of regular topologies that can be built varying the two parameters k and n.

Definition 1 *A fat-tree is a collection of vertices connected by edges and is defined recursively as follows.*

- *A single vertex by itself is a fat-tree. This vertex is also the* root *of the fat-tree.*
- *If $v_1, v_2, \ldots, v_i$ are vertices and $T_1, T_2, \ldots, T_j$ are fat-trees, with $r_1, r_2, \ldots, r_k$ as roots (j and k need not to be equal), a new fat-tree is built by connecting with edges, in any manner, the vertices $v_1, v_2, \ldots, v_i$ to the roots $r_1, r_2, \ldots r_k$. The roots of the new fat-tree are $v_1, v_2, \ldots, v_i$.*

Definition 1 is extremely general and covers many existing examples in the literature. It includes ordinary trees, "full" fat-trees with variable-sized switches and multiple connections between vertices and irregular constructions. The only exception being the orthogonal fat-trees, that allow connections between the roots $v_1, v_2, \ldots, v_i$. Some examples are shown in Figure 2.

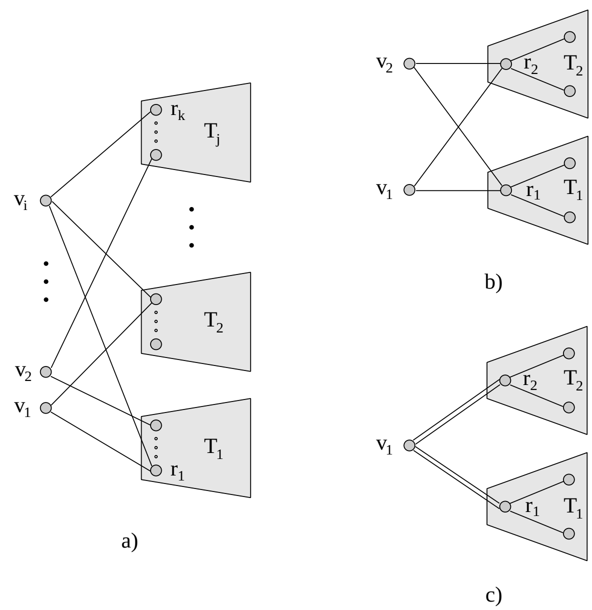

Fig. 2. a) A fat-tree is recursively built connecting the new roots $v_1, v_2, \ldots, v_i$ to the roots $r_1, r_2, \ldots, r_k$ of the subtrees. b) A fat-tree with two roots. c) A fat-tree with multiple edges between the root v_1 and the roots of the subtrees T_1 and T_2.

Let's turn our attention to a particular class of fat-trees, the k-ary n-trees. k-ary n-trees borrow from a popular class of multistage interconnection networks, the k-ary n-butterflies[23] (or k-ary n-flies for short), the topology of the internal switches. The k-ary n-fly is a generalization of the butterfly and is useful to model those topologies that use communication switches with k greater than two. This topology has a recursive structure: one k-ary n-fly contains k butterflies of dimension $n-1$ as subgraphs. Also, each level 0 switch is linked to any level n switch by a unique path of length n, that is k-ary n-flies are *banyan* networks.

We can define the class of k-ary n-trees in the following way.

Definition 2 *A k-ary n-tree is composed of two types of vertices: $N = k^n$ pro-*

cessing nodes and nk^{n-1} $k*k$ *communication switches* [e]*. Each node is an n-tuple* $\{0,1,\ldots,k-1\}^n$*, while each switch is defined as an ordered pair* $\langle w,l\rangle$*, where* $w\in\{0,1,\ldots,k-1\}^{n-1}$ *and* $l\in\{0,1,\ldots,n-1\}$*.*

- *Two switches*

$$\langle w_0,w_1,\ldots,w_{n-2},l\rangle \quad and \quad \langle w'_0,w'_1,\ldots,w'_{n-2},l'\rangle$$

  are connected by an edge if and only if $l'=l+1$ *and* $w_i=w'_i$ *for all* $i\neq l$*. The edge is labeled with* w'_l *on the level* l *vertex and with* w_l *on the level* l' *vertex.*

- *There is an edge between the switch* $\langle w_0,w_1,\ldots,w_{n-2},n{-}1\rangle$ *and the processing node* $p_0,p_1,\ldots,p_{n-1}$ *if and only if*

$$w_i=p_i \quad forall \quad i\in\{0,1,\ldots,n-2\}.$$

  This edge is labeled with p_{n-1} *on the level* $n-1$ *switch.*

It can be easily seen that a k-ary n-tree is a fat-tree, according to Definition 1. In fact, the level 0 switches $\langle w,0\rangle$ are the roots of a k-ary n-tree whose subtrees are $(n-1)$-dimensional k-ary n-trees. Also, the labeling scheme shown in Definition 2 makes the k-ary n-tree a delta network[23]: any path starting from a level 0 switch and leading to a given node $p_0,p_1,\ldots,p_{n-1}$ traverses the same sequence of edge labels $(p_0,p_1,\ldots,p_{n-1})$. Some examples are shown in Figure 3 and Figure 4.

2.1. Topological Properties

k-ary n-trees are built using two building blocks: processing nodes and switches. These switches are logically arranged in a $n*k^{n-1}$ matrix. Communication between a pair of nodes p and q takes place within one of the minimal subtrees of the network that contain both source and destination. The roots of these subtrees can be determined using the numerical representation of both nodes.

Definition 3 *Given a pair of nodes*

$$p=p_0,p_1,\ldots,p_{n-1} \quad and \quad q=q_0,q_1,\ldots,q_{n-1},\ p\neq q,$$

the minimal different index *of* p *and* q*,* $mdi(p,q)$*, is defined as*

$$mdi(p,q) \;=\; \min\{j \mid p_j\neq q_j\}. \tag{1}$$

From the hypothesis that $p\neq q$, we know that there is at least an index j such that $p_j\neq q_j$. The $mdi(p,q)$ is $n-1$ when both nodes are rooted at the same level $n-1$ switch, 0 when p and q belong to distinct k-ary n-trees of dimension $n-1$, and an intermediate value between 0 and $n-1$ otherwise. The mdi can be used to compute the minimal distance between two nodes.

[e] A k-ary n-tree of dimension $n=0$ is composed of a single processing node

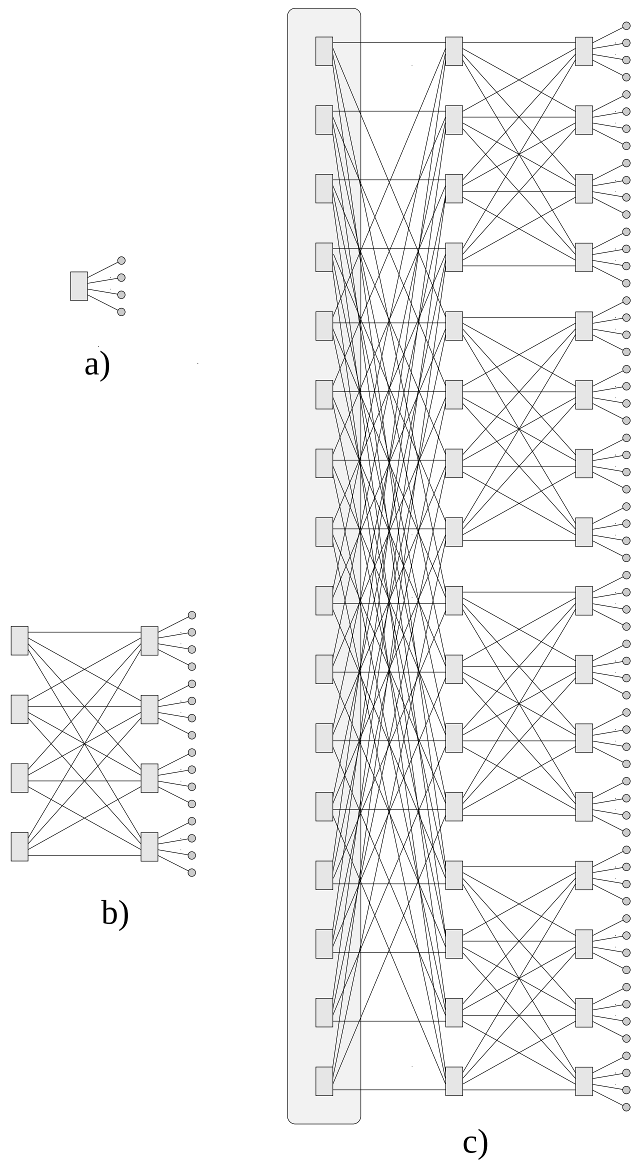

Fig. 3. Some examples of 4-ary n-trees. a) A 4-ary 1-tree is built with a single switch and 4 processing nodes. b) A 4-ary 2-tree has 2 levels with 4 switches for each level. c) Removing the level 0 switches and the corresponding edges of this 4-ary 3-tree we obtain 4 subtrees of dimension 2.

Corollary 1 *The distance between two nodes p and q $p \neq q$, $d(p,q)$, is given by*

$$d(p,q) = 2\left(n - mdi(p,q)\right). \tag{2}$$

Also, we can define the minimal different index of a node and a switch.

Definition 4 *Given a node p and a switch s*

$$p = p_0, p_1, \ldots, p_{n-1}, \quad s = \langle w_0, w_1, \ldots, w_{n-2}, l \rangle,$$

the minimal different index *of p and s, $mdi(p,s)$, is defined as*

- $n-1$ *if* $p_i = w_i,\ i \in \{0,1,\ldots,n-2\}$,
- $\min\{j \mid p_j \neq w_j\}$ *otherwise.*

As in the previous case, the minimal different index between a node and a switch is a number in the range $\{0,1,\ldots,n-1\}$. Using the minimal different index we can characterize the set of nearest common ancestors of any pair of nodes.

Definition 5 *Given two nodes $p = p_0, p_1, \ldots, p_{n-1}$ and $q = q_0, q_1, \ldots, q_{n-1}$, $p \neq q$, with minimal different index $m = mdi(p,q)$, the set of their* nearest common ancestors *$nca(p,q)$ can be defined as*

$$nca(p,q) \;=\; \{\langle p_0, p_1, \ldots, p_{m-1}, w_m, w_{m+1}, \ldots, w_{n-2}, m \rangle\}, \tag{3}$$

with $w_i \in \{0,1,\ldots,k-1\}$.

The cardinality of $nca(p,q)$ is $k^{n-1-mdi(p,q)}$ and varies between 1, when the nodes are directly connected to the same switch, and k^{n-1}. In the second case, $nca(p,q)$ is the set of all level 0 switches.

In the worst case, a message sent from node p to node q must reach one of the level 0 switches and then follow the only path to the destination. This implies that the diameter D of this network is

$$D \;=\; 2n \;=\; 2\log_k N. \tag{4}$$

To compute the average distance d_m we can observe that each node has $k-1$ nodes at distance 2, k^2-k nodes at distance 4, …, $k^n - k^{n-1}$ nodes at distance $2n$. The solution of the corresponding geometric series gives

$$d_m \;=\; 2\left(n - \frac{1}{k-1}\right) + \frac{2}{k^n(k-1)} \;\approx\; 2\left(n - \frac{1}{k-1}\right). \tag{5}$$

This analytic formulation shows that the average distance is very close the network diameter. For example, in the 4-ary n-trees $d_m \approx 2n - 2/3$.

When a k-ary n-tree is divided into two symmetric halves, $k^n/2$ links between level 0 and level 1 switches are cut. Thus, the bisection width B is

$$B \;=\; \frac{k^n}{2} \;=\; \frac{N}{2}. \tag{6}$$

As in the butterflies, the bisection width scales as a linear function of the number of nodes. Also, the overall communication bandwidth between level i and level $i+1$,

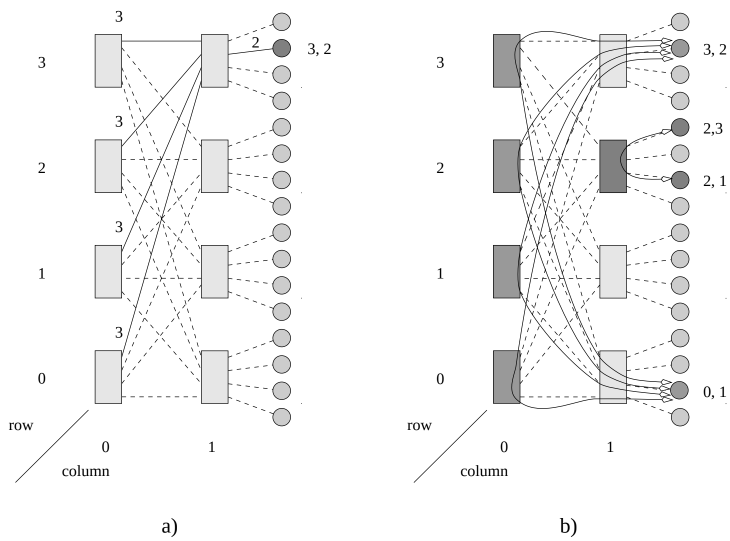

Fig. 4. a) The link labeling of Definition 2 makes the k-ary n-tree a delta network: all paths connecting level 0 switches to the node $3, 2$ (solid lines) cross the same sequence of edge labels $(3, 2)$. b) The *mdi* of nodes $2, 1$ and $2, 3$ is 1: this implies that their *nca* is the common root $\langle 2, 1\rangle$. The *mdi* of nodes $0, 1$ and $3, 2$ is 0; in this case the *nca* is composed of all level 0 switches. These two nodes are connected by four distinct minimal paths.

$i \in \{0, 1, \ldots, n-2\}$ and between level $n-1$ and the processing nodes is constant and proportional to the number of nodes N.

2.2. Message Routing

As outlined above, minimal routing between a pair nodes on a k-ary n-tree can be accomplished sending the message to one of the nearest common ancestors and from there to the destination. That is, each message experiences two phases, an *ascending* phase to get to a nearest common ancestor, followed by a *descending* phase. If we attach at the beginning of each message a header containing the address of the destination $p = p_0, p_1, \ldots, p_{n-1}$, the switches can execute straightforward routing algorithms using the edge labeling scheme of Definition 2. Each switch $s = \langle w_0, w_1, \ldots, w_{n-2}, l\rangle$ has $2k$ edges: k of these are connected to level l–1 switches (if $l > 0$) and the remaining k to level $l + 1$ switches or to processing nodes. We will call, the former, level $l-1$ edges and, the latter, level l edges. The following fragment of pseudo-code describes the skeleton of such a class of routing algorithms.

The routing decision is taken according to the provenience of the message: if it comes from a level l edge, the message is in the ascending phase. The switch reads the message header and computes the $mdi(p, s)$: if it is equal to l, the switch is a nearest common ancestor according to Definitions 3 and 5 and the message can begin the descending phase. The outgoing edge is chosen according to the labeling scheme of Definition 2, in order to correct the l^{th} field. Otherwise, the message continue its ascending path: any of the level $l-1$ edges will do. According to the

```
if ( the message comes from a level l edge )
        /* the message is in the ascending phase */
        if ( mdi(p, s) = l )
                /* begin the descending phase */
                ⟨ route the message to the level l edge labeled p_l ⟩
        else
                /* continue the ascending phase */
                ⟨ pick one of the level l−1 edges ⟩
else
        /* the message is in the descending phase */
        ⟨ route the message to the level l edge labeled p_l ⟩
```

Fig. 5. The skeleton of the routing algorithms.

way we choose this edge we can have different routing algorithms. A *deterministic* algorithm always chooses the same path for a given pair of nodes. A *randomized* algorithm chooses one of these edges according to a pseudo-random function. Unlike randomized algorithms on the cubes, only minimal paths are used to deliver the messages. An *adaptive* algorithm makes a decision according to the local state of the switch, avoiding congested edges. This version is expected to give the best performance, at the cost of a major complexity in the implementation.

Descending messages, coming from a level $l-1$ edge, are routed, according to the usual labeling scheme, to the only path leading to the destination.

The deadlock-freedom[f] can be easily proved building an acyclic buffer or channel dependency graph, according to the flow control strategy in use.[17,10]

It is worth noting that adaptive algorithms on the class of k-ary n-cubes are usually more complicated, when compared to their counterpart for the k-ary n-trees. Several examples in the literature use a large number of virtual channels,[28,9] with sophisticated channel allocation strategies.[5,13] A popular variant of the hot-potato routing based on the exchange protocol, the Chaos routing,[33] requires an internal queue to withdraw messages from the network in the presence of local congestion, making a non trivial routing decision.

3. Relevant Details of the Network Model

This section presents a router model and a simulation environment, that are used in the following sections to analyze the performance of the k-ary n-trees under various traffic loads and flow control strategies.

Figure 6 outlines the internal structure of a $k*k$ routing switch. We can distinguish the external channels or links, the input and the output buffers or lanes that implement the buffer space of the virtual channels and an internal crossbar.

[f] The algorithms are livelock-free because they are minimal and so, they guarantee the progress of each message toward the destination at each routing step.

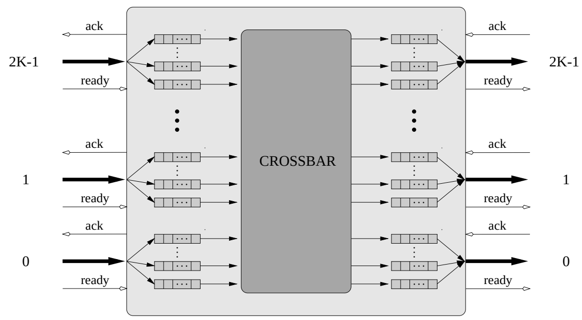

Fig. 6. The internal structure of a routing switch.

The switch has $2k$ bidirectional channels and each channel on the single direction is logically composed of three interfaces: a *data path* that transmits messages on a flit level, the *ready* lines that flag the presence of a flit on the data path and specify the virtual channel where the flit is to be stored and the *ack* lines in the reverse direction that send an acknowledgment every time buffer space is released in the input lanes. The processing nodes have a compatible interface with the same number of virtual channels.

A flit is moved from an output lane to the corresponding input lane in a neighboring node in T_{link} cycles, when there is at least a free buffer position. Each output lane has associated a counter that is initialized with the total number of buffers in the input lane, it is decremented after sending a flit and it is incremented upon receiving and acknowledgment. When multiple lanes are enabled, an arbiter picks one of them according to a fair policy.

When a header flit reaches the top of an input lane, the routing algorithm tries to establish a path in the crossbar with a suitable output lane, that is neither full nor bound to another input lane. This path will remain in action till the transmission of the tail flit of the packet. Our model allows the routing of a single header at a time every $T_{routing}$ cycles. The extra complexity of a parallel router has been shown to give little or no advantage in terms of performance in the presence of wormhole or cut-through flow control.[2] The adaptive routing algorithm studied in this paper, in the ascending phase, picks the channel with the maximum number of free lanes [g].

Although a physical link services in each direction at most one virtual channel every T_{link} cycles, multiple virtual channels can be active at the input and output ports of the crossbar. The internal flit propagation takes $T_{crossbar}$ cycles. Every time a flit is moved from an input lane to the corresponding output lane, a feedback is sent back to the neighboring switch or node to update the counter of free positions.

This model is evaluated in the SMART (Simulator of Massive ARchitectures and Topologies) environment.[34] Implemented in C++, SMART is an object-oriented discrete-event simulation tool for evaluating massively parallel architectures. Configuring some shell scripts, it is possible to select the network topology, the internal router policies and the traffic pattern generated by each node. The simulator allows

[g] A free lane is an output lane not bound to any input lane.

the definition of the packet length, inter-arrival times, number of virtual channels and buffers for both input and output lanes. Also, it is possible to monitor several metrics and time-dependent events, that are gathered in trace files. At the moment, SMART supports three families of topologies: k-ary n-cubes, k-ary n-flies and k-ary n-trees and a node architecture with processing capabilities and a memory hierarchy.

The experiments in this paper evaluate 4-ary 4-trees with 256 processing nodes. The number of virtual channels is varied between 1, 2 and 4 and both input and output lanes have two buffer positions.

Each node generates 20-flit packets with exponentially distributed inter arrival times; the destinations are distributed uniformly or according a static communication pattern, as explained in more detail in the following section. The simulator collects performance data only after 2000 cycles, to allow the network to reach steady state and each simulation is halted after 20000 cycles.

Chien in Ref. [4] has proposed a cost model to make fair comparisons between routing algorithms. This model has gained consideration in several performance studies.[14] It assumes a 0.8 micron CMOS gate array technology for the implementation of the routing chip. The three delays $T_{routing}$ $T_{crossbar}$ and T_{link} are computed as follows.

Routing a message involves address decoding, routing decision and header selection. According to Ref. [4] the routing decision has a delay that grows logarithmically with the number of alternatives, or degree of freedom, offered by the routing algorithm. Denoting by F the degree of freedom, the model estimates the routing delay in

$$T_{routing} = 4.7 + 1.2 \log F \; ns. \tag{7}$$

The time required to transfer a flit from an input channel to the corresponding output channel is the sum of the delay involved in the internal flow control unit, the delay of the crossbar and the set-up time of the output channel latch. The crossbar delay grows logarithmically with the number of ports P. Therefore the crossbar time is

$$T_{crossbar} = 3.4 + 0.6 \log P \; ns. \tag{8}$$

The time required to transmit a flit across a physical link includes the wire delay and the time required to latch it at destination. If virtual channels are used, the virtual channel controller has a delay logarithmic in the number of virtual channels V. The delay of links with medium length wires[15] is estimated by the model in

$$T_{link} = 9.64 + 0.6 \log V \; ns. \tag{9}$$

The values of P and F can be directly computed from the number of virtual channels. The degree of freedom F of a packet in the ascending phase is $(2k-1)*V$, because it can take any of the ascending or descending links and the crossbar size P is $2k*V$. Table 1 summarizes these characteristics and reports the delays of the three flow control strategies. From these results we can see that the 4-ary 4-trees are wire limited and the impact of the virtual channels on the clock cycle is negligible.

Table 1. Characteristics and delays of the three routing algorithms. Delays are expressed in nanoseconds

	V	P	F	$T_{routing}$	$T_{crossbar}$	T_{link}	T_{clock}
1 vc	1	8	7	8.06	5.2	9.64	9.64
2 vc	2	16	14	9.26	5.8	10.24	10.24
4 vc	4	32	28	10.46	6.4	10.84	10.84

In our experiments the delays $T_{routing}$, $T_{crossbar}$ and T_{link} are equalized to a single clock cycle and the clock cycle is the same for the three flow control strategies.

4. Message Generation

In our model each node generates packets independently according to an exponential distribution and the destinations are chosen according to the following traffic patterns. To describe the patterns, let each node $p_0, p_1, \ldots, p_{n-1}$ also be labelled with a number in base k resulting from the concatenation of the p_i. The binary representation of $p_0 p_1 \ldots p_{n-1}$ is $a_0 a_1 \ldots a_{(n \log_2 k)-1}$[h]. Also, let $\overline{0} = 1$ and $\overline{1} = 0$.

- *Uniform traffic.* Destinations are chosen at random with equal probability between the processing nodes.
- *Complement traffic.* Each node sends only to the destination given by $\overline{a_0 a_1 \ldots a_{(n \log_2 k)-1}}$.
- *Bit reversal.* Each node sends only to the destination given by $a_{(n \log_2 k)-1} a_{(n \log_2 k)-2} \ldots a_0$.
- *Transpose.* Each node sends only to the destination given by $a_{(\frac{n}{2} \log_2 k)} a_{(\frac{n}{2} \log_2 k)+1} \ldots a_{(n \log_2 k)-1} a_0 a_1 \ldots a_{(\frac{n}{2} \log_2 k)-1}$.

These traffic patterns illustrate different features. The uniform one is a standard benchmark used in network routing studies. This generation pattern can be considered representative of well-balanced shared memory computations. In the complement traffic all the packets cross the bisection of the network and traverse a path whose length is the diameter. Bit reversal and transpose, are important because they occur in practical computations and can cause worst case behavior in deterministic routers on the class of k-ary n-cubes.[23]

5. Experimental Results

The performance of an interconnection network under dynamic load is usually assessed by two quantitative parameters, the *accepted bandwidth* or *throughput* and the *latency*.

Accepted bandwidth is defined as the sustained data delivery rate given some offered bandwidth at the network input. Two important characteristics are the

[h] We will assume that k is a power of two and n is even.

saturation point and the sustained rate after saturation. Saturation is defined as the minimum offered bandwidth where the accepted bandwidth is lower than the global packet creation rate at the source nodes. It is worth noting that, before saturation, offered and accepted bandwidth are the same. The behavior above saturation is important because the network and/or the routing algorithm can become unstable, leading to a sharp performance degradation. In these cases, a common solution is to send a feedback to the source nodes to reduce the offered bandwidth or to limit the injection rate in the presence of local congestion.[9] We usually expect the accepted bandwidth to remain stable after saturation, both in the presence of bursty applications that require peak performance for a short period of time and applications that operate after saturation in normal conditions, e.g. when executing a global permutation pattern.

Latency is defined as the average delay that a packet experiences during its delivery. We distinguish the following latencies.

- The trailing flit of a packet opens a virtual path between the source and the destination. The *head* latency is the average delay experienced by the trailing flit to get to the network interface of the destination.
- The *tail* latency is the delay needed to absorb the remaining flits of the packet from the network once the trailing flit has arrived to the destination.
- The *network* latency is the average delay spent by a packet in the network, from the insertion of the trailing flit at the source node till the reception of the tail flit at the destination.
- The *end-to-end* latency is the average delay needed to the deliver a packet from a source node to its destination, including queuing delay.

The end-to-end latency rises to infinity above saturation and is impossible to gain any information in this case. For this reason, the network latency is often preferred to analyze the network performance. The network latency, on its turn, is given by the sum of two components, the head and tail latencies.

The experimental results of each traffic pattern are presented according to the Chaos Normal Form[i] (CNF). The CNF uses two graphs, one to display the accepted bandwidth and the other to display the network latency. In both graphs the x-axis corresponds to the offered bandwidth normalized with the unidirectional bandwidth of the links connecting the processing nodes to the network switches. This makes the analysis independent from the link bandwidth and the flit size. Also, it is worth noting that k-ary n-trees are not bisection-bandwidth limited as the k-ary n-cubes, whose CNF is normalized on the bisection bandwidth, the upper bound on the throughput for the uniform traffic.

[i] See http://www.cs.washington.edu/research/projects/lis/chaos/www/presentation.html for more details on the presentation of simulation results of network routing studies.

5.1. Uniform Traffic

Under uniform traffic the adaptive routing algorithm saturates at 36% of the capacity with 1 virtual channel, 55% with 2 virtual channels and 72% with 4 virtual channels, as shown in Figure 7 a). In all cases the post saturation behavior is stable, with a constant throughput for any offered bandwidth. Unlike adaptive algorithms for the k-ary n-cubes that use virtual channels, there are no post saturation problems and no *source throttling* mechanisms[9] are needed to estimate the amount of traffic in the network and limit the injection rate.

These results confirm the importance of the flow control strategy. Wormhole routed networks do not achieve optimal performance in terms of throughput, due to blocking problems. When a packet is stopped at an intermediate switch on the descending phase, all the links on the path[j] from the node/switch where the tail flit is stored to the current switch are blocked. Other packets could profitably use these links. In fact, the use of 4 virtual channels doubles the accepted bandwidth, reaching a considerable 72%. This comes at a price. The condivision of the links between two or more packets slightly increases the network latency for moderate loads. In Figure 7 b) we can see that when the offered load is 20% of the capacity, with 1 virtual channel the network latency is 45 cycles and 49 cycles with 2 and 4 virtual channels. When the load is increased at 30% of the capacity, getting closer to the saturation point of the single virtual channel, the use of more virtual channels pays in term of network latency too: in this case we have 62, 56 and 58 cycles, respectively for 1, 2 and 4 virtual channels.

5.2. Complement Traffic

The CNF of the complement traffic shows a surprising behavior, at least at first glance. As can be seen in Figure 7 c), the saturation point is at about 97% of the capacity for all flow control strategies. This permutation pattern doesn't create any congestion in the descending phase. The use of more than a virtual channel is counterproductive in terms of network latency (Figure 7 d): this is mainly due to the link multiplexing, that increases the tail latency. At steady state, there are as many packets in progress as the number of virtual channels in each link. The network latency with 1 virtual channel remains stable until the offered load is 70% of the capacity and experiences a minor increase of the head latency after this point, which remains under 30 cycles. With 2 virtual channels the head latency has a similar behavior, while the tail latency converges to the upper bound after 70% of the capacity. The network latency with 2 and 4 virtual channels is mainly influenced by the tail latency.

The complement traffic belongs to a wide class of permutations that map a k-ary n-tree into itself. These permutations do not generate any congestion on the descending phase and are called *congestion-free*.[19] The results shown in this section

[j] In our model there are packets with 20 flits and the buffer space at each node is $2+2$ flits in the input and output lanes. In a 4-ary 4-tree for uniform traffic there are, on the average, $6-7$ switches between the source and the destination. For this reason, in the presence of a blocking conflict, it is very likely that the tail flit of the packet is still on (or close to) the source node.

can be generalized to the whole class. In such permutations there are no collisions between packets, because when they are routed on a switch they

- do not require the same output link or
- have at least a channel available in the presence of multiple virtual channels.

Thus, the delays in the head and tail latencies are bounded by the overhead of the flow control strategy[k].

5.3. Bit Reversal and Transpose Traffic

Bit reversal and transpose permutations are often generated by numerical programs and are considered as an interesting benchmark for the interconnection network and the routing algorithm. These permutations have a similar distribution of the destinations in terms of distance. It can be easily noted, looking at the numerical representation shown in section 4, that in both cases there are $k^{n/2}$ nodes at distance 0 (that is source and destination are on the same node) and $(k-1)k^{n/2+i-1}$ nodes at distance $n+2i$, $i \in \{1, \ldots, n/2\}$. The average distance d_m is given by

$$d_m = \frac{k-1}{k^{n/2}} \sum_{i=1}^{n/2} (n+2i)k^i. \tag{10}$$

For a 4-ary 4-tree $d_m = 7.125$, as usual very close to the network diameter. The performance results of these communication patterns are very similar. From the CNF of the bit reversal shown in Figure 7 e) we can see that the saturation points are at 39%, 60% and 78% of the capacity for 1, 2 and 4 virtual channels. An analogous behavior for the transpose can be seen in Figure 7 g), the only difference being the saturation point with 1 virtual channel at 38% of the capacity. After saturation, there is a linear increase of the accepted bandwidth, because there are $k^{n/2}$ nodes sending packets to themselves.

6. Discussion

The congestion-free communication patterns are an important characteristic of the k-ary n-trees. They can be routed reaching optimal performance with a simple routing algorithm and flow control strategy. They are analogous to local communication in direct topologies, as the k-ary n-cubes. The results obtained on the complement traffic generalize to the whole class of congestion-free patterns and are expected to scale with the number of nodes with an accepted bandwidth that approximates the network capacity. Message latency is only influenced by the flow control overhead and can be deterministically estimated with tight upper bounds. Using a sophisticated flow control strategy with several virtual channels is of little help in this case, because increases the network latency.

The remaining communication patterns, uniform, bit reversal and transpose, generate congestion in the descending phase and are very sensitive to the flow control

[k]There are no queuing delays other than those due to channel multiplexing, if any, and the computation of the routing function inside a switch.

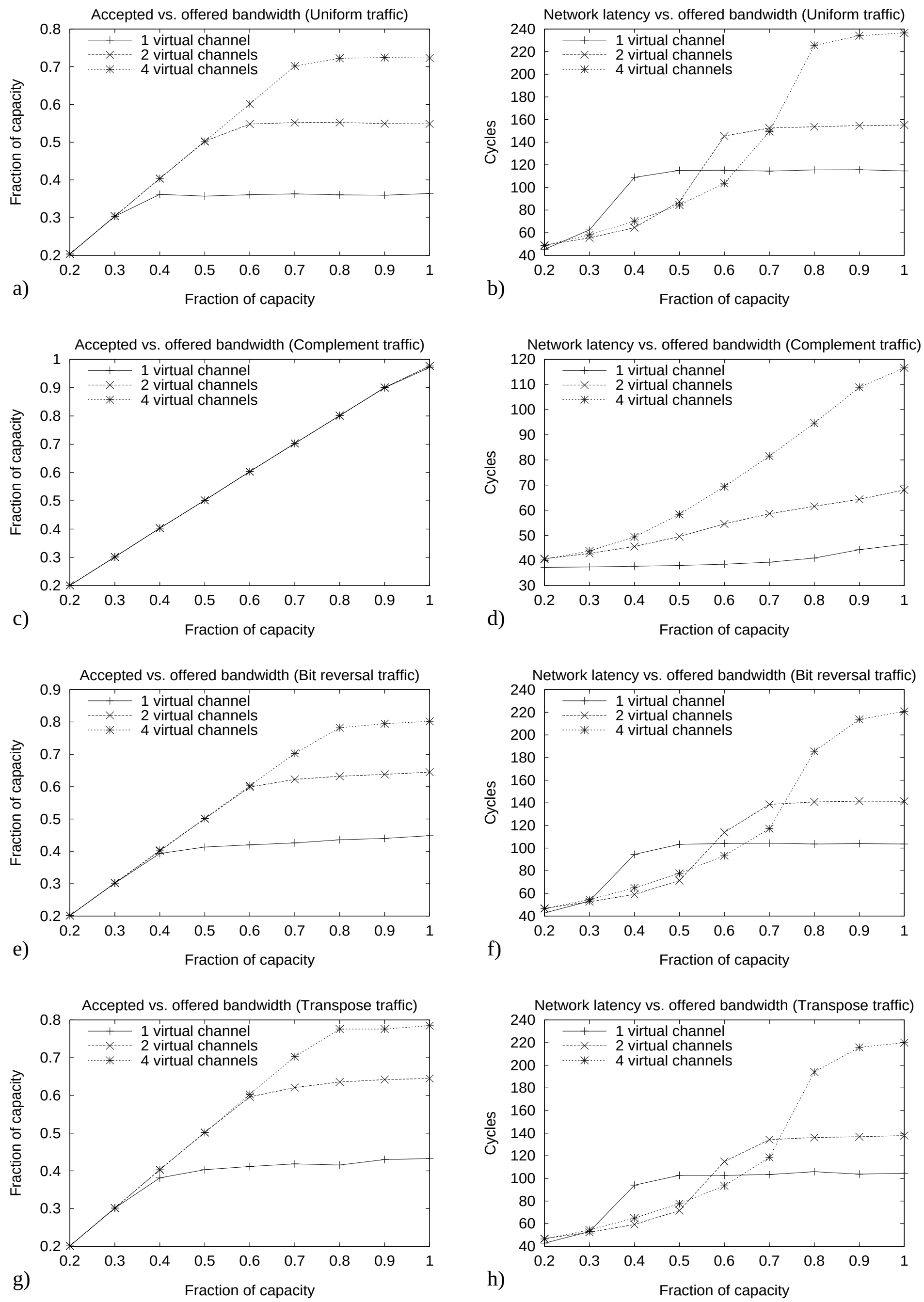

Fig. 7. The Chaos Normal Form (CNF) for the uniform traffic (a-b), complement traffic (c-d), bit reversal traffic (e-f) and transpose traffic (g-h).

Table 2. Saturation points of the communication patterns.

Traffic pattern	*1 vc*	*2 vc*	*4 vc*
Uniform	36	55	72
Complement	97	97	97
Bit reversal	39	60	78
Transpose	38	60	78

strategy. As shown in Table 2, they all saturates at about 35−40% of the capacity with 1 virtual channel, 55−60% with 2 virtual channels and around 75% with 4 virtual channels. From these results we can argue that the expected performance of different communication patterns[l] is mainly influenced by the flow control strategy. Also, in all these cases, switching from 1 to 4 virtual channels doubles the accepted bandwidth. The performance of these patterns is not scalable. In fact, in a 4-ary 3-tree, the saturation points are about 5% above those shown in Table 2 and in a 4-ary 5-tree there is a further decrease of the same amount.

These results provide clear guidelines both for the compilation of parallel programs and for the implementation of the interconnection network. Congestion-free patterns are a powerful tool to reach optimal performance and guaranteed scalability. An important goal for a compiler is to factorize any given application in a collection of congestion-free communication patterns, when possible. In the remaining cases, virtual channels are needed to exploit the network bandwidth and to avoid the blocking problems of wormhole.

7. Conclusion and Future Work

In this paper we have formalized a parametric family of regular topologies, the k-ary n-trees. k-ary n-trees are a particular type of fat-trees, built using processing nodes and constant arity switches interconnected in a butterfly-like topology. We have introduced some of their basic topological properties, including average distance, bisection width and the set of nearest common ancestors of a pair of processing nodes. Message routing on k-ary n-trees can be easily accomplished in two phases. In the ascending phase, each message reaches one the nearest common ancestors of source and destination and then begins the descending phase following the unique downward path to the destination. This framework can be tailored to implement a randomized algorithm, that doesn't increase the average length of the path as in the cubes, or an efficient adaptive algorithm, that simply picks the less loaded link in the ascending phase. The adaptive algorithm has been extensively analyzed on a 4-ary 4-tree with 256 nodes under several traffic patterns, representative of shared memory computations and numerical algorithms. We have compared some variants of the routing algorithm using 1, 2 and 4 virtual channels and the experimental results have outlined two main classes of communication patterns.

- The complement traffic, representative of the class of the congestion-free pat-

[l] Obviously, we are not talking about hot-spot traffic patterns, where the communication load is unevenly distributed on some node or group of nodes.

terns, has reached an optimal performance, with a saturation point at 97% of the capacity for all flow control strategies.

- The remaining traffic patterns, uniform, bit reversal and transpose, are very sensitive to the flow control strategy. The saturation points are between $35-40\%$ of the capacity with 1 virtual channel, $55-60\%$ with 2 virtual channels and around 75% with 4 virtual channels. Also, the introduction of virtual channels has improved the characteristics of the network latency, with tighter upper bounds on the delivery time.

This paper leaves many open questions. In all the patterns the communication load is evenly distributed onto the processing nodes. It would be interesting to study the behavior of the network under hot-spot traffic, to verify how the routing algorithm and the flow control strategy can cope with the tree saturation.[35]

Switching from synthetic traffic to real applications would give a different flavor to the performance results. Generally speaking, this is an open problem not only for the k-ary n-trees, but for many interconnection networks.

Another challenging task is to analyze in depth the class of congestion-free patterns. At the moment we only know some of these patterns and would be interesting to factorize any given communication pattern in a minimal set of congestion-free ones. If this is possible in the general case, we could solve at compile time many important problems, with a simple flow control strategy. Otherwise, the burden is to be left to the use of virtual channels and to a more sophisticated router organization.

References

1. J. D. Allen, P. T. Gaughan, D. E. Schimmel, and S. Yalamanchili, "Ariadne - An Adaptive Router for Fault-tolerant Multicomputers," *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):278–288, April 1994.
2. K. Bolding, *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router*. PhD thesis, University of Washington, Department of Computer Science and Engineering, Seattle, WA, July 1993.
3. K. Bolding, M. L. Fulgham, and L. Snyder, "The Case for Chaotic Adaptive Routing," Technical Report UW-CSE 94-02-04, University of Washington, Department of Computer Science and Engineering, Seattle WA, February 1994.
4. A. A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," In *Hot Inteconnects '93*, Palo Alto, California, August 1993.
5. A. A. Chien and J. K. Kim, "Planar Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors," In *Proceedings of the 19th Annual Symposium on Computer Architecture*, pages 268–277, May 1992.
6. W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Distributed Computing*, 1:187–196, 1986.
7. W. J. Dally, "Virtual Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.
8. W. J. Dally, "Network and Processor Architecture for Message-Driven Computers," In R. Suaja and G. Birthwhistle, editors, *VLSI and Parallel Computers*, chapter 3, pages 140–222. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991.

9. W. J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

10. W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

11. A. DeHon, "Fat-Tree Routing for Transit," Technical Report 1224, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, February 1990.

12. A. DeHon, F. Chong, M. Becker, E. Egozy, H. Minsky, S. Peretz, and T. F. Knight Jr., "Metro: A Router Architecture for High-Performance, Short-Haul Routing Networks," *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):266–277, April 1994.

13. J. Duato, "Deadlock-Free Adaptive Routing Algorithms Multicomputers: Evaluation of a New Algorithm," In *Proceedings of the Third IEEE Symposium in Parallel and Distributed Processing*, pages 840–847, December 1991.

14. J. Duato and P. López, "Performance Evaluation of Adaptive Routing Algorithms for k-ary n-cubes," In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 45–59, Seattle, Washington, USA, May 1994.

15. J. Duato and M. P. Malumbres, "Optimal Topology for Distributed Shared-Memory Multiprocessors: Hypercubes Again?" In *Second International Euro-Par Conference, Volume I*, number 1123 in LNCS, pages 205–212, Lyon, France, August 1996.

16. R. I. Greenberg and C. E. Leiserson, "Randomized Routing on Fat-Trees," *Advances in Computing Research*, 5:345–374, 1989.

17. K. D. Gunther, "Prevention of Deadlocks in Packet-Switched Data Transport Systems," *IEEE Transactions on Communications*, COM-29(4):512–524, April 1981.

18. S. Haridi and E. Hagersten, "The Cache Coherence Protocol of the Data Diffusion Machine," In *PARLE'89, Parallel Architectures and Languages Europe*, volume I, pages 1–18, June 1989.

19. S. Heller, "Congestion-Free Routing on the CM-5 Data Router," In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 176–184, Seattle, Washington, USA, May 1994.

20. Kendall Square Research, *Technical Summary*, 1^{th} edition, 1991.

21. J. H. Kim, Z. Liu, and A. A. Chien, "Compressionless Routing: A Framework for Adaptive Fault-tolerant Routing," *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):289–300, April 1994.

22. T. T. Kwan, B. K. Tatty, and D. A. Reed, "Communication and Computation performance of the CM-5," In *Supercomputing'93*, pages 192–201, November 1993.

23. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, USA, 1992.

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

25. C. E. Leiserson and B. M. Maggs, "Communication-Efficient Parallel Algorithms for Distributed Random Access Machines," *Algorithmica*, 3:53–77, 1988.

26. C. E. Leiserson et al, "The Network Architecture of the Connection Machine CM-5," In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, June 1992.

27. M. Lin, R. Tsang, D. H. C. Du, A. E. Klietz, and S. Saroff, "Performance Evaluation of the CM-5 Interconnection Network," Technical Report AHPCRC Preprint 92-111, University of Minnesota AHPCRC, October 1992.

28. D. H. Linder and J. C. Harden, "An adaptive and Fault-Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

29. A. Martin and D. Bader, "Performance of the CM-5 ENEE 646," unpublished, January 1994.

30. Meiko World Incorporated, *Computing Surface 2 reference manuals*, preliminary edition, 1993.

31. H. L. Muller, P. W. A. Stallard, and D. H. D. Warren, "The Data Diffusion Machine with a Scalable Point-to-Point Network," Technical Report CSTR-93-17, Department of Computer Science, University of Bristol, October 1993.

32. H. L. Muller, P. W. A. Stallard, and D. H. D. Warren, "An Evaluation Study of a Link-Based Data Diffusion Machine," In *Proceedings of the 8th International Parallel Processing Symposium, IPPS'94*, pages 115–128, Cancun, Mexico, April 1994.

33. T. D. Nguyen and L. Snyder, "Performance Analysis of a Minimal Adaptive Router," In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 31–44, Seattle, Washington, USA, May 1994.

34. F. Petrini and M. Vanneschi, "SMART: a Simulator of Massive ARchitectures and Topologies," In *International Conference on Parallel and Distributed Systems Euro-PDS'97*, Barcelona, Spain, June 1997.

35. G. F. Pfister and V. A. Norton, "Hot-spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

36. M. Valerio, L. E. Moser, and P. M. Melliar-Smith, "Using Fat-Trees to Maximize the Number of Processors in a Massively Parallel Computer," In *Proceedings of the 1993 International Conference on Parallel and Distributed Systems*, pages 128–134, Taipei, Taiwan, December 1993.

37. M. Valerio, L. E. Moser, and P. M. Melliar-Smith, "Recursively Scalable Fat-Trees as Interconnection Networks" In *Proceedings of the Thirteenth IEEE International Phoenix Conference on Computers and Communications*, pages 40–46, Phoenix, AZ, April 1994.

38. L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the ACM*, 33(8):103–111, August 1990.