Reducing Complexity in Tree-like Computer Interconnection Networks

Javier Navaridas Jose Miguel-Alonso Francisco Javier Ridruejo Wolfgang Denzel

Department of Computer Architecture and Technology,
The University of the Basque Country, San Sebastian, SPAIN

{javier.navaridas, j.miguel, franciscojavier.ridruejo}@ehu.es

IBM Research GmbH,
Rüschlikon, Switzerland

wde@zurich.ibm.com

ABSTRACT — Interconnection networks based on the *k*-ary *n*-tree topology are widely used in high-performance parallel computers. However, this topology is expensive and complex to build. In this paper we evaluate an alternative tree-like topology that is cheaper in terms of cost and complexity because it uses fewer switches and links. This alternative topology leaves unused upward ports on switches, which can be rearranged to be used as downward ports. The increase of locality might be efficiently exploited by applications. We test the performance of these thin-trees, and compare it with that of regular trees. Evaluation is carried out using a collection of synthetic traffic patterns that emulate the behavior of scientific applications and functions within message passing libraries, not only in terms of sources and destinations of messages, but also considering the causal relationships among them. We also propose a methodology to perform cost and performance analysis of different networks. Our main conclusion is that, for the set of studied workloads, the performance drop in thin-trees is less noticeable than the cost savings.

KEYWORDS: *k*-ary *n*-tree topology, Traffic characterization, Simulation, Performance evaluation

1 INTRODUCTION

The *k*-ary *n*-tree topology [22], based on the classical fat-tree topology introduced by Leiserson [17], is often the topology of choice to build low latency, high bandwidth and high connectivity interconnection networks (hereafter IN) for parallel computers. Its main characteristics are the low mean path length and the multitude of paths from a source to a destination node, which increases exponentially with the distance between nodes (in number of hops). This high path diversity provides a good performance rate for almost all kind of workloads, independently of their spatial, temporal and length distributions.

However, its design does not take into account that parallel applications usually arrange their processes in such a way that communicating processes are as close as possible (in terms of process identifier) to each other, trying to obtain advantages from locality in communication. A network design that ignores locality could be a good option because, in some of the largest parallel systems currently operating, schedulers see processors as an unstructured pool of resources, and assigns them to parallel jobs without guaranteeing that neighbor processes (*i.e.*, consecutive identifiers) run in neighboring compute nodes (attached to the same or adjacent switches). The result is a random

mapping of processes to nodes that may require high bandwidth at all network levels, because many nodes will generate messages addressed to distant. Not all the schedulers function this way: there are a few that are *topology-aware* and schedule applications in consecutive partitions of the network, thus allowing for an effective exploitation of locality. In these cases, for most applications, the bisection bandwidth would no longer be the main performance limiting factor, and the upper levels of network would be under-utilized.

We can reduce cost and complexity of the IN by reducing the ratio between the number of links connected to upper levels and those connected to lower ones. This can be done reducing the radix of the switches or, alternatively, increasing the locality by rearranging the upward ports and making them downward. In both cases the total cost of the system is reduced: fewer switches, fewer links and, in the former case, switches of lower complexity. If parallel applications are correctly placed, performance should not suffer. They could even experience an improvement due to the increased locality of the latter case. In this paper we propose to use thin-trees to, this way, reduce cost and complexity of interconnection networks by doing what we have just described. Thin-trees are directly derived from the *k*-ary *n*-tree topology, reducing the number of upward ports of all switches.

In order to test the different networks, we have performed a throughput study for uniform traffic, both analytically and via simulation. Ideally we would evaluate performance using real traces taken from actual scientific applications running on very large systems but, as large traces are difficult to obtain and not very manageable, we have used a collection of synthetic workloads that emulate their behavior. This mimicry is done not only in terms of spatial patterns, but also in terms of the causality of the injected messages. Some of the communication patterns replicate the way collectives are implemented in common MPI libraries. Others reproduce data interchanges performed in applications that rely on virtual topologies—usually, meshes—commonly used in matrix calculus. The length of the messages and the number of nodes can be specified as parameters.

We have selected some instances of the topologies under study, fed the simulator with the proposed workloads for a variety of message lengths, and measured their performance. A comparison of alternatives is done using raw performance or a performance/cost ratio. As performance is application-dependent, we define a model to compute a performance indicator that can be tailored to fit the characteristics of a given supercomputing center. We will see that, in terms of this indicator, the *k*-ary *n*-tree shows its superiority as a general-purpose topology, although slimmed topologies perform equally well for some relevant application mixes. If cost is considered too, the complexity of the *k*-ary *n*-trees plays against them and the thin-tree is the clear winner: cost is lower and performance is good – in some cases, even better than that of the regular tree, due to a better exploitation of locality.

The rest of this paper is organized as follows. In Section 2 we discuss some topologies in use in former and current high performance computers and also some schedulers and their job placement policies. In Section 3 we present the topologies we will evaluate. The experimental environment—model of the elements, selected topologies and proposed workloads—is explained in Section 4. In Section 5 we show the experimental work and analyze results taking into account only the raw performance. To obtain a fairer comparison of the different topologies, we make a proposal of cost and performance functions, and carry out a performance/cost study in Section 6. We close this work with some conclusions and a future work outlook in Section 7.

2 RELATED WORK

Indirect interconnection networks have evolved noticeably from the first multi-stage networks as those proposed by Clos in [7]. Those networks were built with low-radix switches (typically 4 or 8) and aimed to interconnect at most a few hundred nodes. Current spines, as that on the Mare Nostrum supercomputer [5], have switches with hundreds of ports and are able to interconnect thousands of nodes. Former trees were low-radix: the CM-5 [18] had a radix-8 data network. Current ones use switches with higher radices, as those radix-24 of the Cray XD1 [9]. There are also recent tree-like proposals as the Black Widow Clos network [11] that takes advantage of the high availability of ports (radix-64 switches) to add side-links to the common tree-like arrangement. However, the most noticeable change in these networks is that former indirect networks were built *ad hoc* for the target systems, whereas current high-performance networking technologies as QsNet [21], Myrinet [19] or InfiniBand [15] have favoured building super-clusters with *off-the-shelf* components.

Network bandwidth and latency have experienced notable improvements during the last 10 years, from the 800Mbps of the ASCI Red (1997) [26] to the 20Gbps currently available in InfiniBand [15] when using 4X, dual-data-rate connections, or the 10Gbps by Myri-10G and 10Gb Ethernet, both offered by Myricom [19]. Soon we will see offers of 100-120 Gbps (100G Ethernet, InfiniBand 12X-QDR). This takes us to a network bandwidth improvement over 100 times in 10 years. The latency of the full protocol and the network in the ASCI Red (taking into account message passing library) is 12μs. Both Myri-10G and InfiniBand latencies are around 2μs. Thus, latency has been improved (around 6 times), but not as noticeably as bandwidth has.

Taking a look at the most current Top500 list [12], we can see two clear trends. On the one hand, the choice of topology for *custom-made*, massively parallel computers is the 3D cube. On the other, *commodity-based* systems (super-clusters) are built around the class of trees discussed in this paper. Most of the machines in the middle positions are arranged this way, which justifies our interest in tree-like topologies.

We stated in Section 1 that common schedulers do not take into account the underlying network topology. The only supercomputer we have found that tries to maintain locality is the BlueGene family (3D tori), whose scheduler [3] puts tasks from the same application in one or more mid-planes (8x4x4). On the contrary, the scheduling strategy [1] of Cray XT3/XT4 family (also 3D tori) gets the first available compute processors. Some of the other job queuing and scheduling managers like Sun Grid Engine [27], Load Leveler [14], or MOAB [8] do not offer locality-aware politics, but provide mechanisms to implement them.

3 TOPOLOGIES UNDER STUDY

In this section we will describe two different multi-stage, tree-based topologies. In these descriptions we assume that all switches used to build a given network have the same radix. For the purpose of this paper we leave unplugged the upward ports of the topmost level of switches. This assumption has advantages in terms of simplicity

in the descriptions, and also provides scalability. The disadvantage is in terms of cost, because some resources are unused; this is particularly relevant for those topologies with more switches in the top level. In practical implementations, all ports of the highest switch level may be used as downward ports, eventually resulting in a larger network size. Alternatively, we may consider a single switch as an aggregation of lower radix *virtual* switches, which results in a smaller number of switches in the topmost stage of the system.

3.1 Definitions

In the graphical representations of the topologies (see Fig. 1), boxes represent switches and lines represent links between them. Note that we show neither the compute nodes connected to the first level switches and their links, nor the last level of upward links (which, as we stated before, are unplugged). These elements are hidden for the sake of clarity.

Throughout this paper we will use n to denote the number of levels in a network, and N to denote the number of compute nodes (leaves) attached to it. We will denote the total number of switches in a topology as S, and the number of switches at level i as S_i. The total number of links will be denoted as L. The switch radix will be denoted as R. L_B will denote the link bandwidth, B_B the bisection bandwidth, and B_C the number of channels in the bisection. We will denote the theoretical, ideal throughput for uniform traffic as Θ. We call the relation between the number of downward ports of a switch and the number of upward ports the *slimming factor*. For example, taking a look at the switches in the topology shown in Fig. 1b, four ports are downward ports, linked to switches in the next lower level. The remaining two ports of each switch are upward ports that connect to switches in the next higher level; therefore the slimming factor is 2:1.

In the topological descriptions that follow, we denote each switch port within the system as the level where the switch is, the position of the switch in that level, and the number of the port in that particular switch. We call the lower level of switches (those attached to compute nodes) level *0*; obviously, level *n-1* is the one on the top of the tree. We number the switches in each level from left to right, starting from *0*. Ports in a switch are denoted as upward (↑) or downward (↓), and numbered from left (*0*) to right. Thus, a port can be addressed as a 4-tuple <*level*, *switch*, *port*, *direction*>.

Given two ports P and P', they are linked ($P \leftrightarrow P'$) when there is a connection (link) between them. As links are full-duplex, in the expressions concerning linkage we avoid the redundancy of showing downward connections. We will call *level*, *switch* and *port* the address components of a given port, and *nlevel*, *nswitch* and *nport* the address components of the port to which it is connected (its upper neighbor). Therefore,

$$< level, switch, port, \uparrow > \leftrightarrow < nlevel, nswitch, nport, \downarrow >$$

Along this paper, we will refer to *heavy* and *light* workloads. Light workloads are those in which the number of messages circulating simultaneously through the network is low, and the length of the messages is short. In contrast, heavy workloads are those in which most of the nodes are injecting messages at once so that the network will experience peaks of congestion; this situation would be worse if messages are addressed to distant destinations.

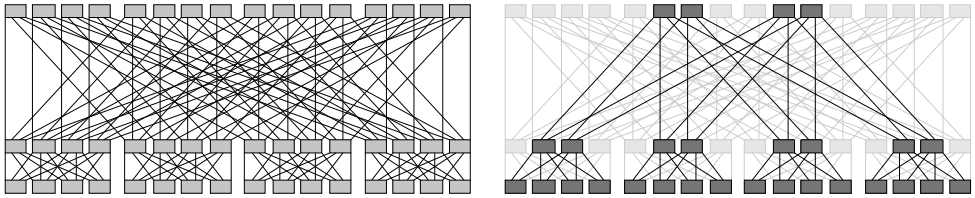


Fig. 1. Samples of the topologies under study to build 64 nodes networks. a) 4,3-tree. b) 4:2,3-thin-tree.

3.2 *k*-ary *n*-tree

This is the best-known of the topologies considered in this study. It will be the yardstick to compare the thin-tree against. *k*-ary *n*-trees [22], where *k* is half the radix of the switches—actually, the number of links going upward (or downward) from the switch—and *n* the number of levels, will de denoted through this paper as *k:k,n*-tree. Note that in this case the slimming factor is 1:1.

A *k*-ary *n*-tree is typically built in a butterfly fashion between each two contiguous levels, Fig. 1a shows a depiction of a 4-ary 3-tree. The topological neighborhood description is as follows:

$$\forall level \in [0,\ n-1),\ \forall switch \in [0,\ k^{n-1}),\ \forall port \in [0,\ k):$$

$$nlevel = level + 1$$

$$nswitch = \left(\left(port \cdot k^{level} \right) + \left(k^{level-1} \cdot \left\lfloor \frac{switch}{k^{level}+1} \right\rfloor \right) + \left(switch \bmod k^{level} \right) \right) \bmod k^{n-1}$$

$$nport = \left(\left\lfloor \frac{switch}{k^{level}} \right\rfloor - \left(k^{level+1} \cdot \left\lfloor \frac{switch}{k^{level}} \right\rfloor \right) \right) \bmod k$$

The main advantages of this topology are the high bisection bandwidth and the large number of routing alternatives for each pair of source and destination—a path diversity that can be exploited via adaptive routing. Nevertheless, they might be expensive and complex to deploy, because of the large number of switches and links.

Note that bandwidth remains constant in all levels. Most parallel application exhibit some level of locality in communication. This means that, in actual scenarios, the higher the level, the lower is the utilization of resources in that level. This is what supports the utilization of slimming strategies: an attempt to reduce complexity in the upper levels without sacrificing application performance.

3.3 *k*:*k'*-ary *n*-thin-tree

We define a thin-tree as a *cut-down* version of a *k*-ary *n*-tree in which we apply a given slimming factor. We will denote them as *k*:*k'*,*n*-tree, being *k* the number of downward ports, *k'* the number of upward ports and *n* the number of levels. The slimming factor is, obviously, the ratio between *k* and *k'*. *k* does not need to be a multiple of *k'* so that we can produce a thin-tree with arbitrary values of *k* and *k'*. It is remarkable that a *k*-ary *n*-tree is actually a *k*:*k*-ary *n*-thin-tree.

A 4:2-ary 3-thin-tree is depicted in Fig. 1b. Removed switches and links from a full-fledged *k*-ary *n*-tree are shaded. The topological neighborhood relationship between ports in a thin-tree is described as follows:

$$\forall level \in [0,\ n-1),\ \forall switch \in \left[0,\ k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{n-level}\right),\ \forall port \in [0,\ k'):$$

$$nlevel = level + 1$$

$$nswitch = \left(\left(port \cdot k'^{level}\right) + \left(switch \bmod k'^{level}\right) + \left\lfloor \frac{switch}{k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{level}} \right\rfloor \cdot k' \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{level} \right)$$

$$nport = \left\lfloor \frac{switch}{\left\lfloor \frac{k}{k'} \right\rfloor^{level}} \right\rfloor \bmod k$$

In this topology the bisection bandwidth has been reduced, as well as the number of switches and links (*i.e.*, cost and complexity). We want to investigate how applications suffer this reduction. Thin-trees are easier to deploy than regular trees and, if k and n values are kept, the radix of switches is smaller.

3.4 Theoretical Throughput

We open this sub-section with Table 1, which summarizes the relations between network parameters (n, k and k'), number of elements (N, S, S_i, L, R) and topological properties (B_B, B_C, Θ) for the topologies under study. The computation of Θ for both topologies—a common way to evaluate interconnection networks performance—is done in the following paragraphs, via an analytical study.

As the real throughput of a system depends on the link bandwidth, we will focus the study on the accepted load relative to the maximum load a node is able to inject. To do this, we assume that the link bandwidth is constant trough the whole network. This way, as every node is connected to the network via a single link, the throughput should be in (0, 1]. Note that zero-throughput means the absence of connectivity.

TABLE 1.
TOPOLOGICAL CHARACTERISTICS OF THE TOPOLOGIES

	***k*-ary *n*-tree**	***k*:*k'*-ary *n*-thin-tree**
Nodes	$N = k^n$	$N = k^n$
Switches	$S = n \cdot k^{n-1}$	$S = \sum_{i=0}^{n-1} k'^{(n-i)-1} \cdot k^i$
Switches per level $\forall i \in [0, n-1]$	$S_i = k^{n-1}$	$S_i = k^{(n-i)-1} \cdot k'^i$
Links	$L = S \cdot k$	$L = S \cdot k$
Switch radix	$R = 2 \cdot k$	$R = k + k'$
Channels in Bisection	$B_C = \frac{k^n}{2}$	$B_C = \frac{k'^{(n-1)} \cdot k}{2}$
Bisection Bandwidth	$B_B = \frac{k^n}{2} \cdot L_B$	$B_B = \frac{k'^{(n-1)} \cdot k}{2} \cdot L_B$
Theoretical Throughput	$\Theta = 1$	$\Theta = \left(\frac{k'}{k}\right)^{n-1}$

As stated in chapter 3 of [10] the upper boundary of the throughput of a given topology could be expressed as the ratio between the bisection bandwidth and the number of nodes. Therefore, the theoretical ideal relative throughput (hereafter ideal throughput) is:

$$\Theta = \frac{2 \cdot B_B}{N \cdot L_B} \tag{1}$$

As the link bandwidth is constant, the number of channels in the bisection is calculated as:

$$B_C = \frac{B_B}{L_B} \tag{2}$$

From (1) and (2) we obtain:

$$\Theta = \frac{2 \cdot B_c}{N} \tag{3}$$

In both cases, the number of channels in the bisection is half the number of links at the last stage, *i.e.*:

$$B_C = \frac{S_{n-1} \cdot k}{2} \tag{4}$$

This way, we can compute the ideal throughput for uniform traffic as:

$$\Theta = \frac{2 \cdot S_{n-1} \cdot k}{2 \cdot N} \tag{5}$$

Thus, from (5) and looking at the values of S_i in Table 1, the ideal throughput of a k-ary n-tree is:

$$\Theta = \frac{k^{n-1} \cdot k}{k^n} = 1 \tag{6}$$

And the ideal throughput of a *k:k'*-ary n-thin-tree is:

$$\Theta = \frac{k'^{(n-1)} \cdot k}{k^n} = \left(\frac{k'}{k}\right)^{n-1} \tag{7}$$

4 Experimental Set-Up for Simulation-Based Evaluation

We use INSEE [23] to evaluate some different tree-like networks, feeding them with a collection of application-inspired synthetic workloads. The simulator measures time in terms of cycles, the time required by a phit to traverse one switch (switching plus transmission).

4.1 Switches

For this work, we have used simple input-buffered switches whose radices range from 9 to 16, depending on the topology. In order to keep things simple, we do not use virtual channels, except if explicitly indicated. The arbitration of each output port is performed in a random way, that is, every time an output port is free it randomly chooses

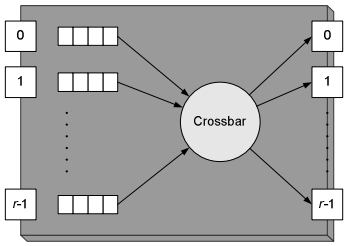


Fig. 2. Model of the switch used in the simulations given a radix *r*.
Ports at the left are input ports and those at the right are output ports.

among all the input ports that have requested this resource. Transit queues are located in the input ports and are able to store 4 packets. A schematic model of the switch is depicted in Fig. 2.

In this work we model the node as a traffic generation source with one injection queue, which is able to store 8 packets. It is also the sink of the arrived messages. When generating traffic, we consider reactive sources, meaning that the reception of a message may *trigger* the release of a new one. This way we can model the causality inherent to actual application traffic. Messages are split into packets of a fixed size of 16 phits. One phit is the smallest transmission unit, fixed to 128 bits. If a message does not fit exactly in an integral number of packets, the last packet contains unused phits.

The switching strategy is virtual cut-through. Routing is adaptive (but restricted to shortest paths in order to avoid deadlock) using a credit-based mechanism, being the credit the space in the queue of the neighbor's input port. This mechanism works as follows: when several output ports are feasible, the one with more available credits (more room in the neighboring queue) is selected, if several ports have the same amount of credit, one of them is selected at random. Credits are communicated out-of-band, so they do not interfere with regular traffic. The use of adaptive routing allows taking the best of each of the topologies, which in turn allows for a fairer comparison among them. Reader should note that optimal static routing functions are application-dependent and have established a line of research by themselves. An interesting contribution that is in line with the contents of this paper is the static routing algorithm presented in [25] which was used with the kind of trees studied in this paper.

4.2 Networks under Study

In this work we have performed three sets of evaluations for the topologies under study. In the first evaluation, which revolves around the theoretical throughput, we have fixed the number of downward ports per switch (set to 8), the slimming factors (set to 8:6, 8:4 and 8:2) and the target number of connectable compute nodes (set to 4096). With these restrictions, we have worked with the following topologies: 8:8,4-tree, 8:6,4-tree, 8:4,4-tree and 8:2,4-tree.

The second set of experiments used traffic from some kernels of applications to feed a wide variety of networks, all of them using switches with 8 downward ports. All the possible slimming factors have been used, from 8:8 (the complete tree) to 8:1. Futhermore we have tested three different scales of the system, able to connect 64 nodes (2 levels), 512 (3 levels) and 4096 nodes (4 levels). All the evaluated topologies and some of their characteristics are summarized in Table 2.

TABLE 2.
CHARACTERISTICS OF THE TOPOLOGIES IN THE SECOND SET OF EXPERIMENTS

a)	8:1,2-tree	8:2,2-tree	8:3,2-tree	8:4,2-tree	8:5,2-tree	8:6,2-tree	8:7,2-tree	8:8,2-tree
Nodes	64	64	64	64	64	64	64	64
Switchs	9	10	11	12	13	14	15	16
Radix	9	10	11	12	13	14	15	16
Links	72	80	88	96	104	112	120	128
b)	**8:1,3-tree**	**8:2,3-tree**	**8:3,3-tree**	**8:4,3-tree**	**8:5,3-tree**	**8:6,3-tree**	**8:7,3-tree**	**8:8,3-tree**
Nodes	512	512	512	512	512	512	512	512
Switchs	73	84	97	112	129	148	169	192
Radix	9	10	11	12	13	14	15	16
Links	584	672	776	896	1032	1184	1352	1536
c)	**8:1,4-tree**	**8:2,4-tree**	**8:3,4-tree**	**8:4,4-tree**	**8:5,4-tree**	**8:6,4-tree**	**8:7,4-tree**	**8:8,4-tree**
Nodes	4096	4096	4096	4096	4096	4096	4096	4096
Switchs	585	680	803	960	1157	1400	1695	2048
Radix	9	10	11	12	13	14	15	16
Links	4680	5440	6424	7680	9256	11200	13560	16384

a) 64-node systems. **b)** 512-node systems. **c)** 4096-node systems.

As we have just stated, all the switches have 8 downward ports. However, the actual radix of the switches is not always the same, being smaller in the more slimmed topologies. Thus, in these two evaluations the complete tree has advantage compared with the thinner alternatives: it uses more links, and more switches that also are larger. Thus performance measurements are biased towards the 8:8,4-tree.

In the last evaluation set we have fixed the radix of the switches (set to 12), and used all the feasible slimming factors (from 11:1 to 6:6). Under these restrictions we have created the smallest topologies capable to connect at least 64, 512 and 4096 nodes. The result of the evaluations would be fairer than in the previous set, because all the switches have the same radix. Note how thinner topologies have lower bandwidth and path diversity than regular trees, but in return locality is increased. Unfortunately, the proposed networks have different sizes, in terms of the number of compute nodes they can connect. As we are using workloads that emulate a fixed number of tasks, we are not capable of using all the trees' leaves. Table 3 summarizes some characteristics of the networks in this third evaluation set. For example note how it is possible to build a 8:4,4-thin-tree with exactly 4096 nodes, but the complete 6:6,5-tree built with 12-port switches has 7776 leaf-nodes. In terms of cost this plays against the topologies that provide the worst fit to the target number of nodes.

TABLE 3.
CHARACTERISTICS OF THE TOPOLOGIES IN THE THIRD SET OF EXPERIMENTS

a)	11:1,2-tree	10:2,2-tree	9:3,2-tree	8:4,2-tree	7:5,3-tree	6:6,3-tree
Nodes	121	100	81	64	343	216
Switchs	12	12	12	12	109	108
Radix	12	12	12	12	12	12
Links	132	120	108	96	763	648
b)	**11:1,3-tree**	**10:2,3-tree**	**9:3,3-tree**	**8:4,3-tree**	**7:5,4-tree**	**6:6,4-tree**
Nodes	1331	1000	729	512	2401	1296
Switchs	133	124	117	112	888	864
Radix	12	12	12	12	12	12
Links	1463	1240	1053	896	6216	5184
c)	**11:1,4-tree**	**10:2,4-tree**	**9:3,4-tree**	**8:4,4-tree**	**7:5,5-tree**	**6:6,5-tree**
Nodes	14641	10000	6561	4096	16807	7776
Switchs	1464	1248	1080	960	4440	5184
Radix	12	12	12	12	12	12
Links	16104	12480	9720	7680	31080	31104

a) 64-node systems. **b)** 512-node systems. **c)** 4096-node systems.

4.3 Workloads

First of all we fed the networks under study with synthetic uniform traffic with independent sources to perform a *classic* throughput evaluation. This set of experiments will show us a first estimation of the networks' potential, and their capacity to handle heavy workloads. It also will validate the analytical study performed in Section 3.4.

As we stated in the introduction, we would also like to test the selected networks with *realistic* traffic, ideally taken from traces obtained from applications running in actual supercomputers. Since it is difficult to obtain and handle traces of applications running on thousands of nodes, we decided to create instead some synthetic traffic generators which emulate data interchanges typically used in scientific parallel applications.

The patterns of choice will be described in the following paragraphs. In the descriptions that follow, N is the number of processes in the parallel application, identified from 0 to N-1. Note that we assume a mapping of one process to one compute node attached to one leave of the network, which results in the consecutive allocation of the tasks into the nodes—*i.e.* task n goes to node n. The graphical representation of these patterns is depicted in Fig. 3. In this depiction, blue arrows and squares represent processes, likely, black arrows represent messages. Note that the circle-end of the arrow represents a message sent, the arrow-end represents that the node must stall until the reception. The time flows from left to right.

For all patterns each node starts with an initial set of messages that have to be injected into the network. The length of these messages is configurable. Messages must be packetized before injection, thus long messages generate a burst of packets. The reception of a message may *trigger* the release of some additional ones; this means that patterns include *causal relationships*. In the simulator, we start with an empty network and measure the time used to consume the initial collection of messages, plus all additional messages generated by causal relationships, until the network is empty again.

We have generated these patterns for fixed network sizes of N=64, N=512 and N=4096 compute nodes and message lengths of 40KB—as will be explained later, in waterfall pattern the size of the actual messages is much smaller. Most of the patterns in use in this work were further defined and justified in [20].

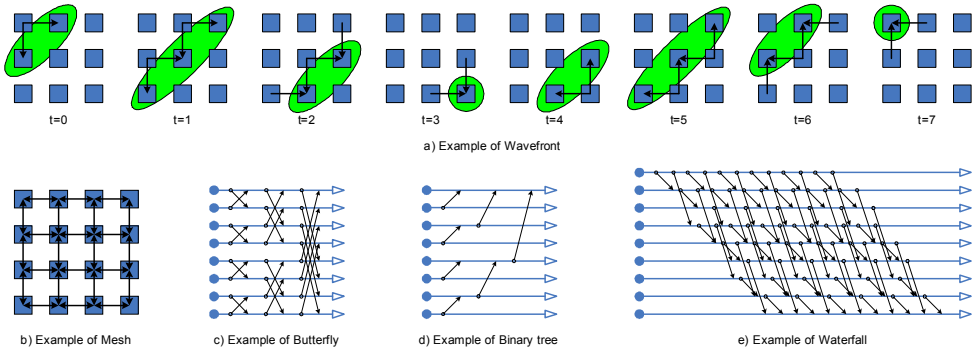


Fig. 3. Graphical representation of the traffic patterns: Time flows from left to right. Blue lines and squares represent nodes. Each black arrow start means a message send. The end of the black arrows means that the node has to stop until receiving the corresponding message. a) Wave-front (Green) in a 3x3 2D-mesh. b) Neighbor interchange in a 4x4 2D-mesh. c) Butterfly that emulates N-to-N collectives (8 nodes). d) Binary tree that emulates N-to-1 collectives (8 nodes). e) Waterfall pattern (9 nodes).

The 2D and 3D wave-front patterns (**W2** and **W3**) perform a diagonal *sweep* from the first node to the last one in MPI virtual square (or cubic) meshes, and then returns to the first node. The simulation of this patterns starts with two (three for **W3**) messages in node 0, and ends with the finalization of the return sweep. These patterns are considered light—note that there are only a few nodes injecting at once—but create some contention in the destination nodes because they have to receive data from several neighbors. We can observe this pattern in applications implementing the Symmetric Successive Over-Relaxation (SSOR) [6] algorithm—used to solve sparse, triangular linear systems.

The 2D and 3D mesh patterns (**M2**, **M3**) perform data movements in MPI virtual square (or cubic) meshes from every node to all its neighbors; after that, each node waits for the reception of all messages from its neighbors. Simulation starts with all nodes injecting one message per direction (2-4 for **M2**, 3-6 for **M3**), and ends once all messages arrive. These patterns impose a very heavy load on the network, because all nodes inject simultaneously several messages at once before stopping to wait for the receptions. These patterns can be observed in applications using finite difference methods [2].

The butterfly pattern (**BU**) provides an efficient implementation of MPI N-to-N collectives (MPI_Alltoall, MPI_Allreduce, etc.) [28]. It is also known as "recursive doubling". Simulation of **BU** starts with a message at each node, and ends when defined by the pattern. This is a heavy pattern because all the nodes inject at once, and also in the last stages of the butterfly the messages have to traverse the whole network, which may exhacerbate congestion.

The binary tree pattern (**BT**) provides an efficient implementation of some N-to-1 MPI collective operations, such as MPI_Reduce and MPI_Gather [16]. Simulation of this pattern starts with a message at odd-numbered nodes, and ends when node 0 receives the messages from all power of two nodes (included 2^0=1). This is the lightest of the patterns because there is almost no contention in the delivery of the messages.

The waterfall traffic pattern (**WF**) is inspired in a pattern we have observed in the NAS Parallel Benchmark LU [24]. It consists of a large collection of small messages, with causal dependencies. For this pattern, we define a total number of bytes to transmit (*length*) and, instead of starting with a single 40KB message; a burst of 40 messages of 1KB length is generated. **WF** can be seen as a burst of **W2**s (actually, LU uses SSOR) but using small messages of fixed length. Node 0 starts a burst of messages that flood the network. The simulation ends when the last burst arrives to node *N*-1. The main characteristic of **WF** is the presence of causality chains. Latencies in the delivery of messages accumulate at the end of the chain. We can consider this pattern heavy because during the execution mean time, most of the nodes are injecting messages at once, however it is not as heavy as **BU** or **M2** and **M3** because the causality chains throttle the injection of messages.

5 Experiments and Analysis of Results

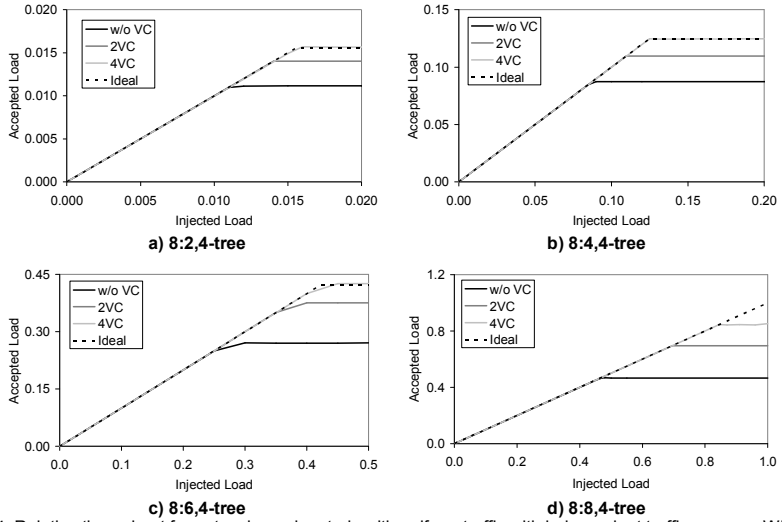
In this section we will evaluate the networks describe above. In the initial set of experiments we feed the networks with uniform traffic from independent traffic sources. These preliminary, non-realistic tests could make us think that

thin-trees are not a good topological choice. However, a deeper study using a selected mix of workloads that mimic applications behavior tells us a different story: when taking into account that high performance applications' processes synchronize and maintain causal relationships, the load that traverses the last stages of a tree-like topology is smaller than that traversing the first stages; therefore, we can build trees that are thinner at the upper stages without adversely affecting the execution time of applications—but with a very positive impact in terms of budget.

5.1 Throughput

The first set of experiments to evaluate the performance of the topologies under test will be by means of a classical throughput measurement for uniform traffic from independent traffic sources. We will plot the accepted load *versus* the offered load to, this way, have a first estimation of how fast the networks reach saturation and how much performance could we expect from them under heavy loads. We perform these measurements via simulation using the following methodology. First of all, there is a fixed *warm-up* period of 30K cycles. After that, a *convergence phase* is started. In this phase we measure the average load every 1K cycles. When four consecutive measurements are within a range of 5%, we consider that the convergence phase is finished and that the simulation has reached a *stationary phase*. After this point, the throughput measurement is started. This is the last phase in which 10 batches of 5K cycles each are captured. The result shown is the average accepted load of these ten batches. Note that the standard deviation of the average accepted load of the batches in all experiments is lower than the 0.5% of the average value.

We plot in Fig. 4 the throughput evaluation for the four networks under study. As stated in Chapter 13 of [10], when using virtual channel flow control the average throughput of the network is increased due to contention reduction. Thus, in order to compare and validate the analytical study in sub-section 3.4, we also plot the results for



a) 8:2,4-tree **b) 8:4,4-tree**

c) 8:6,4-tree **d) 8:8,4-tree**

Fig. 4. Relative throughput for networks under study with uniform traffic with independent traffic sources. With 1, 2 or 4 VC and the upper boundary of the throughput. Note the difference in the axis ranges of each topology.

the same topologies using 2 and 4 virtual channels (denoted as 2VC and 4VC respectively) and the computed ideal throughput curve.

We can observe that the *k*-ary *n*-tree topology is not able to reach the ideal limit, not even when using 4 VC. Note that the ideal limit of 1 means that all nodes send and receive one phit per cycle which is the maximum allowed by any link attaching a node to the network. Any form of contention, including that at destination nodes, will reduce the actual throughput. The utilization of multiple VCs can help reducing unnecessary contention, but it cannot be completely eliminated. In the case of the thin-trees (8:6,4-tree 8:4,4-tree and 8:2,4-tree), nodes can still inject up to one phit per cycle, which is much higher that the theoretical throughput limit. Consumption is never a bottleneck, and the utilization of a few virtual channels allows the network to reach the ideal throughput.

The reader should note the difference in the axis range for the three experiments. This is because the ideal throughput for the networks under study is 1 for the 8,4-tree, ~0.42 for the 8:6,4-tree, 0.125 for the 8:4,4-tree and ~1.6E-2 for the 8:2,4-tree.

This first performance evaluation say us that thin-trees are not a good idea if we have to deal with random traffic from independent sources. However, in the next subsections we will study the networks with workloads that *mimic* actual parallel applications.

5.2 Experiments with Same Size Networks

Comentario [y1]:

In this set of experiments we have gathered the time (in simulation cycles) used by the networks to deliver all the messages of each of the application-kernels. As these times differ widely, due to the characteristics of the patterns, we have normalized them, using the times for the complete trees as the reference. These normalized times are represented in Fig. 5.

The reader can observe that most thin-tree networks perform acceptably well in light traffic patterns (**BI**, **W2** and **W3**). With them, the *k*-ary *n*-tree cannot take advantage of its high bandwidth and path diversity, just because the network occupancy is low and so is the probability of two packets competing for an output port in a switch. In contrast, under heavy loads, the high bandwidth of the *k*-ary *n*-tree topology is able to handle the high amount of packets inside the network. In the slimmed networks, still, there is too much contention due to the bandwidth reduction between each level so the packet delivery is slower. This is especially noticeable in the large-scale configuration in which delivery of the messages may suffer a slowdown of up to 41.

Regarding the slimming factors, the topologies with the 8:7 slimming factor show delays in therms of the delivery times that were always below 13% and in most cases below 5%. 8:6 slimmed topologies were always below 21% and in most cases below 10%. 8:5 slimmed topologies were always below 50% and in most cases below 30%.The 8:4 slimmed topologies are showed to be the inflection point of performance, the slowdown values of the networks with more aggressive slimming factors rocket to values that may be intolerable.

Obviously, the smaller is the network, the less noticeable is the effect of thinning the topology, but, as we will see later, the reduction in terms of cost are also smaller. It is also noticeable that the behaviour of the networks does not scale with the size of the network.

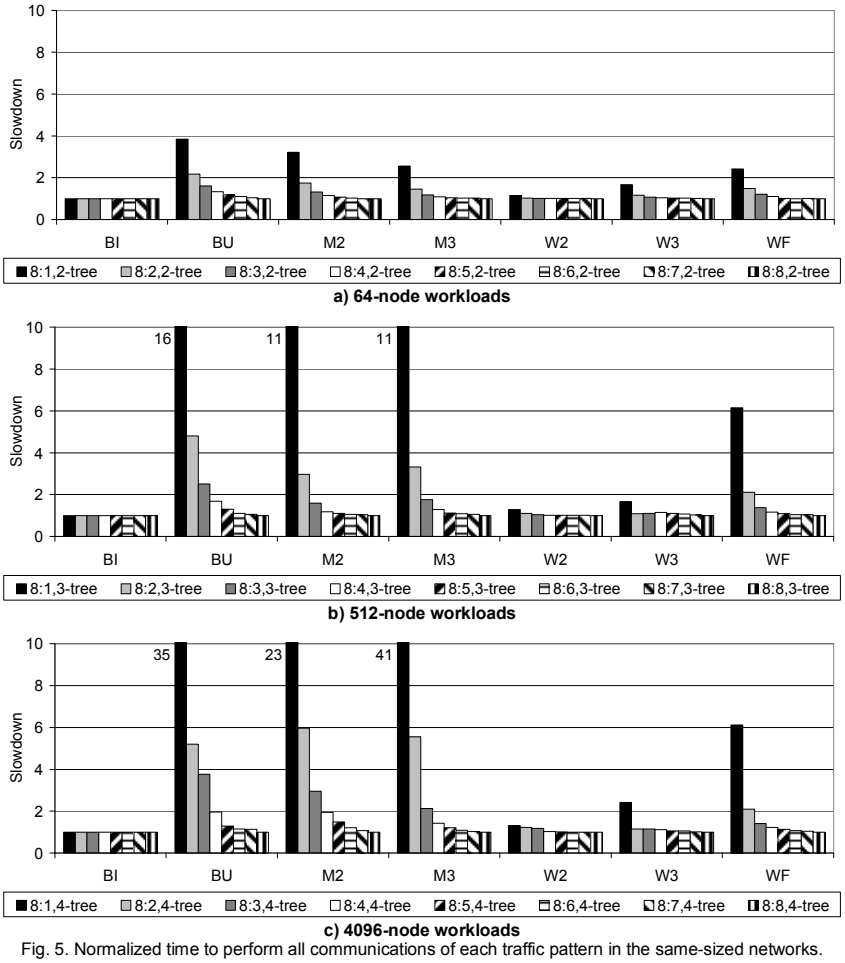


Fig. 5. Normalized time to perform all communications of each traffic pattern in the same-sized networks.

We can conclude from this set of experiments that the *k*-ary *n*-tree is the best-performing topology in almost all experiments (combinations of pattern and message length). Nonetheless, in many cases thin-trees with reduced slimming factors (up to 8:5 or 8:4, depending on the workload) performs equally well. Additional slimming causes excessive network contention, so results are noticeably worse (over 40 times in the largest configuration).

5.3 Experiments with Same Radix Switches

In this subsection we analyze the results of the experiments comparing topologies built with switches with the same radix (12 ports). Results are plot in Fig. 6. In general, they confirm what we learned from the previous set of experiments. Nevertheless, in this case, the slimmed topologies have an additional advantage: the increased ability to exploit locality in communication. A switch in a 6,5-tree uses 6 ports as downward ports, and 6 ports as upward ports. In contrast, the same switch in a 8:4,4-thin-tree has 8 downward and 4 upward ports. In other words, in the

slimmed topologies the unused upward ports are rearranged to work as downward ports. To a certain extent, this compensates the reduction of links and switches in the upper levels. The result is that slimmed topologies may outperform the complete trees. For example, the **BU** pattern is one of the heaviest workloads, and the 8:4,2-tree is able to deliver it in less time than the 6:6,3-tree.

The **W2** and **W3** patterns require specific attention. Note the excellent performance of thin-trees. The high causality of this pattern does not allow the utilization of all the resources of the complete trees, but allows for a productive exploitation of additional levels of locality—for instance, 7:5,5-tree and 8:4,4-tree deliver both workloads faster than the 6:6,5-tree.

Again, the slimming factor should not go very far. The performance of the thinnest topologies (9:3,4-tree and thinner) is too low for the heavier workloads, reaching slowdowns of up to 90 times in the largest configuration.

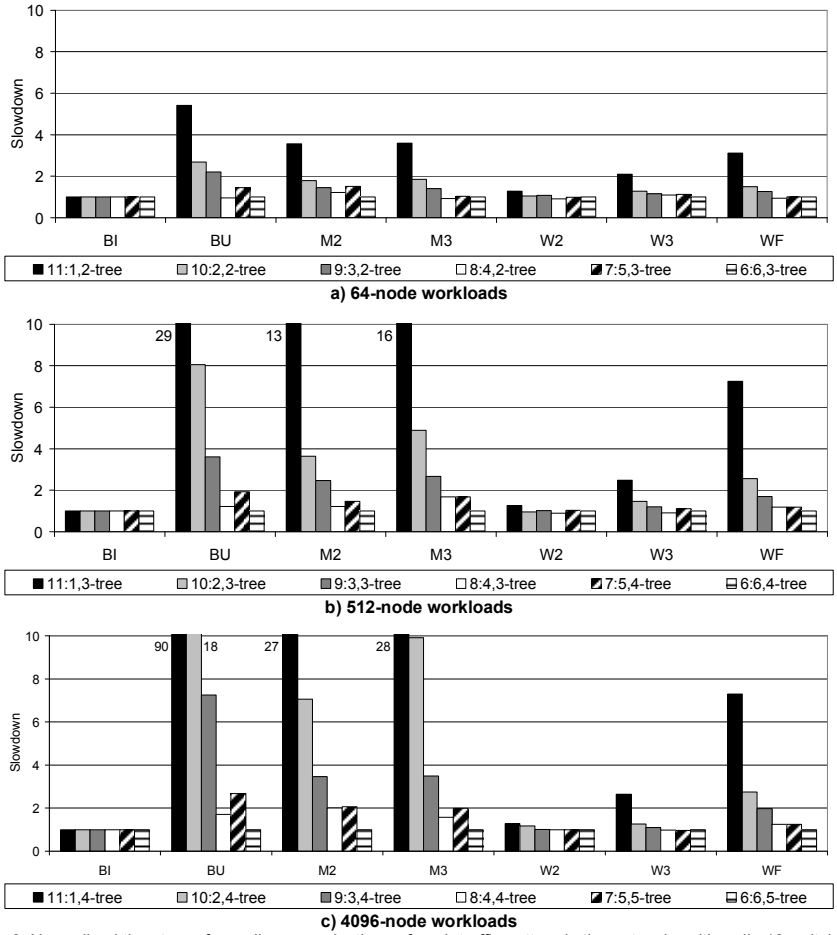


Fig. 6. Normalized time to perform all communications of each traffic pattern in the networks with radix-12 switches.

To summarize this section, we can state that for the lighter workloads, there is almost no difference between topologies—again, ignoring cost. For heavy loads, networks with 1:2 slimming factor, perform as well as, or even better, than those with 1:1. Growing up to higher slimming factors is counter-productive.

6 PERFORMANCE/COST ANALYSIS

In the previous section we have carried out a comparison of topologies taking into account only their raw performance. We have ignored the costs of the networks under evaluation. These costs differ widely from network to network, and *must* be taken into account if we want to make a fair comparison. We propose a methodology to compare the networks. It takes into account that it is necessary to measure the performance of the network by evaluating it with appropriate workloads.

6.1 Characterizing Performance

If we had unlimited (financial) resources we could just select the *best-performing* option, but that option may not be the most *cost-effective*. Here we propose a means to measure the effectiveness of a network that takes into account the workloads using it.

Actual workloads vary widely from site to site, depending on the applications in use. In this work we are not using actual applications, but a collection of synthetic—but representative—workloads. We describe a network-efficiency function in the context of these workloads that can be extended with further workload types.

For each given workload simulation reported a (relative) time T_W. For example, we have a certain execution time T_{BU} for butterfly. Note that these values are relative to the complete trees: consequently they are always 1 for this topology. Depending on the application mix of interest in a particular computing center, we may apply a weighting factor to each workload w_W. This weight should be large for those applications that are used often. For a given network, we define its performance φ as follows:

$$\varphi = \frac{1}{\sum_W w_W \cdot T_W}$$

Note that for a given application mix (and a set of weights) a higher value of φ represents a better-performing network. As in this work we can neither identify all representative application mixes nor even use actual applications, we decided to use a constant value of one for all the weights, with the solely purpose of illustrating the

TABLE 4. PERFORMANCE OF THE SAME SIZE NETWORKS

a)	φ	b)	φ	c)	φ
8:1,2-tree	0.4419	**8:1,3-tree**	0.1410	**8:1,4-tree**	0.0628
8:2,2-tree	0.6970	**8:2,3-tree**	0.4272	**8:2,4-tree**	0.3157
8:3,2-tree	0.8354	**8:3,3-tree**	0.6746	**8:3,4-tree**	0.5164
8:4,2-tree	0.9094	**8:4,3-tree**	0.8273	**8:4,4-tree**	0.7243
8:5,2-tree	0.9539	**8:5,3-tree**	0.9088	**8:5,4-tree**	0.8569
8:6,2-tree	0.9791	**8:6,3-tree**	0.9523	**8:6,4-tree**	0.9276
8:7,2-tree	0.9900	**8:7,3-tree**	0.9695	**8:7,4-tree**	0.9647
8:8,2-tree	1.0000	**8:8,3-tree**	1.0000	**8:8,4-tree**	1.0000

a) 64-node systems. **b)** 512-node systems. **c)** 4096-node systems.

TABLE 5. PERFORMANCE OF THE SAME RADIX NETWORKS

a)	φ	b)	φ	c)	φ
11:1,2-tree	0.3491	**11:1,3-tree**	0.0980	**11:1,4-tree**	0.0441
10:2,2-tree	0.6277	**10:2,3-tree**	0.3105	**10:2,4-tree**	0.1667
9:3,2-tree	0.7341	**9:3,3-tree**	0.5135	**9:3,4-tree**	0.3633
8:4,2-tree	0.9969	**8:4,3-tree**	0.8653	**8:4,4-tree**	0.7359
7:5,3-tree	0.8644	**7:5,4-tree**	0.7453	**7:5,5-tree**	0.6411
6:6,3-tree	1.0000	**6:6,4-tree**	1.0000	**6:6,5-tree**	1.0000

a) 64-node systems. **b)** 512-node systems. **c)** 4096-node systems.

proposed methodology. With this constant weight, the denominator in our efficiency value is just the addition of the (relative) times obtained in the experiments. This yields a value of φ=1/7 for the *k*-ary *n*-tree. We further normalize this value to be in the range [0, 1]. Table 4 and Table 5 show the normalized performance values for the two sets of experiments. Note how, using this criterion, the best performing networks are the complete trees, however some configurations of thin-trees have similar φ value.

6.2 Cost of the Networks

Performing an exhaustive cost analysis of a complete system is, clearly, a difficult task that requires the knowledge of a large number of parameters, including the choice of technologies and physical placement of the elements of the system (nodes, racks). A proper cost evaluation should take into account both deployment and maintenance costs. Deployment cost must consider the number of switches and links, that may, and probably will, have different characteristics—for instance, the use of wires of different length would be needed for most plant organizations. Maintenance costs include the power consumption and the heat dissipation.

At any rate, all this concerns are outside of the scope of this paper. For this reason, we will consider three simple functions to compute the cost of each network in order to be able to carry out a performance/cost comparison. Note that these functions bear in mind some different aspects of the design of a system and, consequently the actual cost function may be a mixture of these three.

In these functions, *S* represents the total number of switching elements of the network and *R* their radix. Note that in our topological model the upward ports of the topmost stage are unplugged, and therefore the last levels of the trees are formed by switches with a smaller radix. However, for the sake of simplicity, we will consider that all the switches have the same radix. Furthermore, to simplify these functions, the number of links is not taken into account as it depends on the number of switches and ports.

The considered cost functions are the following:

- In the first function c_C, the cost of the switch is constant regardless of its radix, $c_C = S$. Several aspects of the manufacture of the network scale linearly with the number of switches independently of their radix, such as the cost related to the plant area, the rack space or the packaging of the switch.
- In the second function c_L, the cost of the switch depends linearly on the radix, $c_L = S \cdot R$. For instance the the number of links scales linearly with the radix, as well as the cost of the hardware associated to each port.

- In the third function c_Q, the cost increases quadratically with the radix, $c_Q = S \cdot R^2$. Note that the heart of a switch (the crossbar) scales quadratically with the number of ports [13].

Using the characteristics of the networks previously introduced in Table 2, we have computed the cost of the same size systems using each of the three cost functions. Furthermore, using the performance figures shown in Table 4 we have shown the cost-efficiency of the systems, calculated as the performance divided by the cost, and normalized to that of the complete tree. All these values are collected in Table 6. Note that measuring the cost of the same radix networks would be a bit tricky—and probably unfair—because every configuration has a different number of nodes. For this reason we will restrict the evaluation of the performance/cost efficiency to the same size networks.

For the small-scale configuration, we can see how the performance/cost efficiency of the 8:2,2-tree is the highest for the linear and quadratic cost functions, being 8:3,2 the most efficiency when considering the constant cost function. This is because its performance is good, but the cost is reduced almost to one half compared with the complete tree. In this case all the configurations have better efficiency than the complete tree regardless of the used cost function. The only exception is the 8:1,2-tree with the constant cost function, whose loss in terms of performance is not compensated by the reduction in number of switches.

In the case of the medium-scale configuration, the 8:4,3-tree is the best contender when using the constant cost function, while the 8:3,3-tree wins for the other two cost functions. In this case, not all the thinned topologies are able to beat the complete tree, 8:1,3-tree only does for the quadratic cost function.

If we focus on the large-scale configuration, we can notice that things are not as clear as before. The 8:4,4-tree is the most cost-effective for the linear and constant cost functions, but its performance is reduced considerably, around 30% lower than that of the complete tree. The 8:5,4-tree may be a better option due to a noticeable boost in terms of performance, even when its cost-efficiency is not as high as the 8:4,4-tree's. Finally, it is remarkable that, even an

TABLE 6. PERFORMANCE/COST EFFICIENCY OF THE SAME SIZE NETWORKS

a)	**8:1,2-tree**	**8:2,2-tree**	**8:3,2-tree**	**8:4,2-tree**	**8:5,2-tree**	**8:6,2-tree**	**8:7,2-tree**	**8:8,2-tree**
c_C	9	10	11	12	13	14	15	16
φ/c_C	0.7857	1.1152	**1.2152**	1.2125	1.1740	1.1190	1.0560	1.0000
c_L	81	100	121	144	169	196	225	256
φ/c_L	1.3967	**1.7843**	1.7675	1.6166	1.4449	1.2789	1.1264	1.0000
c_Q	729	1000	1331	1728	2197	2744	3375	4096
φ/c_Q	2.4831	**2.8549**	2.5709	2.1555	1.7783	1.4616	1.2015	1.0000

b)	**8:1,3-tree**	**8:2,3-tree**	**8:3,3-tree**	**8:4,3-tree**	**8:5,3-tree**	**8:6,3-tree**	**8:7,3-tree**	**8:8,3-tree**
c_C	73	84	97	112	129	148	169	192
φ/c_C	0.3708	0.9765	1.3352	**1.4182**	1.3527	1.2354	1.1014	1.0000
c_L	657	840	1067	1344	1677	2072	2535	3072
φ/c_L	0.6592	1.5623	**1.9421**	1.8910	1.6648	1.4119	1.1748	1.0000
c_Q	5913	8400	11737	16128	21801	29008	38025	49152
φ/c_Q	1.1719	2.4997	**2.8249**	2.5213	2.0490	1.6136	1.2531	1.0000

c)	**8:1,4-tree**	**8:2,4-tree**	**8:3,4-tree**	**8:4,4-tree**	**8:5,4-tree**	**8:6,4-tree**	**8:7,4-tree**	**8:8,4-tree**
c_C	585	680	803	960	1157	1400	1695	2048
φ/c_C	0.2197	0.9509	1.3171	**1.5452**	1.5167	1.3569	1.1656	1.0000
c_L	5265	6800	8833	11520	15041	19600	25425	32768
φ/c_L	0.3906	1.5214	1.9158	**2.0602**	1.8668	1.5508	1.2434	1.0000
c_Q	47385	68000	97163	138240	195533	274400	381375	524288
φ/c_Q	0.6944	2.4342	**2.7866**	2.7470	2.2976	1.7723	1.3262	1.0000

a) 64-node systems. **b)** 512-node systems. **c)** 4096-node systems.

awful-performing configuration, the 8:2,4-tree, with a measured performance of 0.32, overtakes the complete tree when using the quadratic cost function.

The reader should note that the network is only a part of the system, so that the execution time depends (greatly) on the behavior of the other components, and the interactions between all of them. In other words, the advantages or disadvantages of a given network might not be as clear as shown in our evaluations. This is an argument against the better-performing, more-expensive networks, because in real set-ups the benefit of using them will be diluted. The extent of this dilution depends on the applications and their data-sets, as well as on the architecture of the system. Furthermore, the collection of workloads used in this analysis might not be representative—and probably is not—of all actual workloads used at supercomputing centers. A thorough study should be customized for a particular site, taking into account their applications and fine-tuning their relative weights.

7 CONCLUSIONS AND FUTURE WORK

In this paper we have described and characterized a slimmed version of *k*-ary *n*-trees: the *k*:*k'*-ary *n*-thin-trees. A thin-tree can be seen as a *k*-ary *n*-tree after removing some links and switches. A thin-tree costs a fraction of the price of a complete tree, in terms of deployment and maintenance. In terms of performance, thin-trees with low slimming factor perform as well as comparable *k*-ary *n*-trees. Excessive slimming (beyond 2:1 in our workbench) results in bad performance results. A qualitative comparison of performance/cost ratios turns out to be very favorable for the thin-trees in the studied cases.

Thin-trees are obviously cheaper than the complete tree, but their bandwidth in the upper levels is greatly reduced. After removing links and switches in the upper levels, performance can be maintained (or even increased) due to an effective exploitation of locality. Using fixed-radix switches, a thin-tree devotes more ports to downward links, so more nodes can communicate without using the upper levels.

At least three super-clusters that are in positions #1 (RoadRunner), #41 (Tsubame) and #70 (Thunderbird) of the June 2009 edition of this list were built with InfiniBand networks arranged as thin trees—actually, narrowed spines. In Thunderbird the slimming factor is 2:1; RoadRunner and Tsubame goes further, to 4:1 and 5:1 respectively. We have not found any evaluation work providing the rationale behind those decisions. However, in this work we have shown that, compared to full-fledged *k*-ary *n*-trees, thinner topologies provide comparable performances at much lower cost. The savings in the number of networking elements could be invested in other ways to improve the system (faster CPUs, better performing networking technology, enlarge the system, etc)

This study opens several lines for future work. We want to further explore the workload generation mechanism, to make it mimic the characteristics of parallel applications as accurately as possible. One interesting place to start is [4], were 13 “dwarfs” are identified as representative scientific applications (an extension of the original “seven dwarfs” introduced by Phil Colella).

The *scheduling* problem is also of interest. The advantage of slimmed topologies is obtained through an efficient exploitation of locality—something that is impossible to achieve under the flat-network assumption in commonly used schedulers. As in the case of the BlueGene [3], we need to make scheduler's topology-aware.

ACKNOWLEDGMENT

This work has been supported by the Ministry of Education and Science (Spain), grant TIN2007-68023-C02-02, and by grant IT-242-07 from the Basque Government. Mr. Javier Navaridas is supported by a doctoral grant of The University of the Basque Country.

REFERENCES

[1] R Ansaloni, "The Cray XT4 Programming Environment". Available at: www.csc.fi/english/csc/courses/programming_environment

[2] Y Aoyama, J Nakano. "RS/6000 SP: Practical MPI Programming". IBM Red Books SG24-5380-00, Chapter 4, p.99-153, ISBN 0738413658. August, 1999. Available at: http://www.redbooks.ibm.com/abstracts/ sg245380.html

[3] Y Aridor et al. "Resource allocation and utilization in the Blue Gene/L supercomputer". IBM J. Res. & Dev. Vol. 49 No. 2/3 March/May 2005. Available at: http://www.research.ibm.com/journal/rd/492/aridor .pdf

[4] K Asanovic et al. "The Landscape of Parallel Computing Research: A View from Berkeley". EECS Department. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. December 18, 2006. Available at: http://www.eecs.berkeley.edu/Pubs/TechRpts/ 2006/EECS-2006-183.pdf

[5] Barcelona Supercomputing Center "Mare Nostrum". Available at: http://www.bsc.es/

[6] E Barszcz, R Fatoohi, V Venkatakrishnan, and S Weeratunga, "Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors", Technical Report NAS RNR-93-007, NASA Ames Research Center, Moffett Field, CA, 94035-1000, April 1993.

[7] C Clos: "A Study of Non-Blocking Switching Networks" Bell System Technical Journal, March 1953, pp.406-424.)

[8] Cluster Resources. "Moab Workload Manager Administrator's Guide". Available at: http://www.clusterresources.com/moabdocs/MoabAdminGuide510.pdf

[9] Cray Inc., "Cray XD1 Overview". Available at: http://www.cray .com/ products/xd1/

[10] WJ Dally, B Towles. "Principles and Practices of Interconnection Networks". Morgan-Kaufmann, 2004.

[11] WJ Dally et al., "The BlackWidow High-Radix Clos Network", Proceedings of the 33rd annual international symposium on Computer Architecture, p.16-28, 2006, June 17-21.

[12] JJ Dongarra, HW Meuer, E Strohmaier. "Top500 Supercomputer sites". Nov. 2007 edition. Available at: http://www.top500.org/

[13] H El-Rewini, and M Abd-El-Barr. "Advanced computer architecture and parallel processing" Wiley, 2005. ISBN: 978-0-471-46740-3

[14] IBM. "IBM LoadLeveler for AIX 5L: Using and Administering". Available at: http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/ IBMp690/IBM/usr/lpp/LoadL/html/am2ugmst02.html

[15] Infiniband Trade Association. "Infiniband® Trade Asociation". Available at: http://www.infinibandta.org

[16] S Labour, "MPICH-G2 Collective Operations, Performance Evaluation, optimizations". Available at: http://www-unix.mcs.anl.gov/ ~lacour/argonne2001/report.ps

[17] CE Leiserson. “Fat-trees: Universal networks for hardware efficient supercomputing”. IEEE transactions on Computers, C-34(10):892–901, October 1985.

[18] CE Leiserson et al., “The Network Architecture of the Connection Machine CM-5”, Symposium on Parallel Algorithms and Architectures (April 1992).

[19] Myricom. “Myrinet home page”. Available at: http://www.myri.com/

[20] J Navaridas, J Miguel-Alonso, FJ Ridruejo. "On synthesizing workloads emulating MPI applications". The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-08). April 14-18, 2008, Miami, Florida, USA.

[21] F Petrini, W Feng, A Hoisie, S Coll and E Frachtenberg. “The Quadrics Network: High-Performance Clustering Technology”. IEEE Micro 22, 1 (Jan. 2002), 46-57. DOI= http://dx.doi.org/10.1109/40.988689

[22] F Petrini and M Vanneschi. “k-ary n-trees: High Performance Networks for Massively Parallel Architectures”. In Proceedings of the 11th International Parallel Processing Symposium, IPPS’97, p. 87–93, Geneva, Switzerland, 1997.

[23] F.J. Ridruejo, J. Miguel-Alonso. "INSEE: an Interconnection Network Simulation and Evaluation Environment". Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005), Pages 1014 - 1023.

[24] F. J. Ridruejo, J. Navaridas, J. Miguel-Alonso and C Izu, "Realistic Evaluation of Interconnection Network Performance at High Loads", 8th Int. Conf. on Parallel and Distributed Computing Applications and Technologies - PDCAT 2007, Adelaide, Australia, 3-6 December 2007.

[25] G Rodriguez, R Beivide, C Minkenberg, J Labarta and M Valero. “Exploring pattern-aware routing in generalized fat tree networks”. Procs of the 23rd international conference on Supercomputing, Yorktown Heights, NY, USA, 2009, pp. 276-285. DOI: 10.1145/1542275.1542316

[26] Sandia National Labs, “ASCI Red”. Available at: http://www.sandia .gov/ASCI/Red/

[27] Sun Microsystems, Inc. “N1 Grid Engine 6 User’s Guide”. Available at: http://docs.sun.com/app/docs/coll/1017.3

[28] R Thakur and W Gropp, “Improving the Performance of Collective Operations in MPICH”, Available at: http://www-unix.mcs.anl.gov/ ~thakur/papers/mpi-coll.pdf