Reducing Complexity in Tree-like Computer Interconnection Networks

Techical Report
EHU-KAT-IK-06-07
Dep. of Computer Architecture and Technology, UPV/EHU

Javier Navaridas Jose Miguel-Alonso Francisco Javier Ridruejo Wolfgang Denzel
Dep. of Computer Architecture and Technology, UPV/EHU, Spain IBM Research, Rüschlikon, Switzerland
javier.navaridas@ehu.es j.miguel@ehu.es franciscojavier.ridruejo@ehu.es wde@zurich.ibm.com

Abstract

The fat-tree is one of the topologies most widely used to build high-performance parallel computers. However, they are expensive and difficult to build. In this paper we propose two alternative tree-like topologies that are cheaper in terms of cost and complexity, because they are "thinner" than fat-trees: less switches, and less links. We test the performance of these narrowed trees, and compare it with that of the full-fledged fat-tree using a collection of synthetic patterns that emulate the behavior of well-known applications, including causal relationships among messages. Moreover, we do a performance/cost analysis of these networks. Our main conclusion is that, for the set of studied applications, performance of narrowed topologies is as high as that of fat-trees, with much less cost. Rearranging switch ports to use them as downward ports instead of upward ports do reduce bisection bandwidth, but also increases locality, which can be efficiently exploited by applications.

Reducing Complexity in Tree-like Computer Interconnection Networks

Javier Navaridas Jose Miguel-Alonso Francisco Javier Ridruejo Wolfgang Denzel
Dep. of Computer Architecture and Technology, UPV/EHU, Spain IBM Research, Rüschlikon, Switzerland
javier.navaridas@ehu.es j.miguel@ehu.es franciscojavier.ridruejo@ehu.es wde@zurich.ibm.com

1 Introduction

The k-ary n-tree topology [9], also known as fat-tree, is often the topology of choice to build low latency, high bandwidth and high connectivity interconnection networks 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. This high availability of paths provides a good performance rate for almost all kind of workloads, independently of their spatial, temporal or size distributions.

However, the fat-tree 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 process identification) to each other, trying to obtain advantages from locality in communication. A design that ignores locality could be a good option because, in some of the biggest parallel systems currently operating, schedulers see processors as an unstructured pool of resources, and assigns them to parallel jobs without guaranteeing that neighbor processes (e.g., consecutive identifiers in MPI applications) run in neighbor compute nodes (attached to the same or adjacent switches). The result is a random mapping of processes to nodes that may require a high bisection bandwidth at all network levels, because many nodes will generate messages addressed to distant nodes, and the larger the distance the larger the number of network levels traversed. 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. For most applications, the bisection bandwidth of the whole network is then not as important as before. Thus, when using these "smart" schedulers the upper levels of the fat-tree topology are under-utilized.

We can reduce the fat-tree cost and complexity by reducing the proportion between the number of links connected to upper levels and those connected to lower ones. This way, we can choose between reducing the radix of the switches or, alternatively, increasing the locality by rearranging the upward ports and making them downward ones. In both cases the total cost of the system is reduced: less switches, less links and, in the former case, switches of lower complexity. If parallel applications are correctly placed, performance should not suffer. It could even increase due to the increased locality of the latter case.

We propose in this paper two narrowed trees with reduced cost and complexity by doing what we have just described. The first proposal is directly derived from the fat-tree, reducing the number of upward ports of all switches. In contrast to the full fat-tree we call this topology *thin-tree*. The second proposal is a novel topology that allows traffic to adapt not only in the upward path – as in fat-trees – but also in the downward path, with the purpose of reducing contention in the second part of the trip of messages. A drawback of this topology is the number of levels: it is taller (meaning it has a greater number of levels) than fat-trees and thin-trees. For this reason we have called it *slender-tree*. In addition to describing the characteristics of these topologies, we define a cost model for them.

To test the different topologies, we have not performed the usual throughput and latency tests using random traffic or permutations. Ideally we would like to use real traces 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 realized in MPI implementations. 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 them 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. In terms of this indicator, the fat-tree shows its superiority as a general-purpose topology, although narrowed topologies perform equally well for some patterns. If cost is considered too, the complexity of fat-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 fat-tree, due to a better utilization of locality.

The rest of this paper is arranged as follows. In Section 2 we present the fat-tree topology and the narrowed proposals. The experimental environment – model of the elements, selected topologies and proposed workloads – is explained in Section 3. In Section 4 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 performance and cost functions, and do a performance/cost study in Section 0. We close this work with some conclusions and a future work outlook in Section 6.

2 Topologies Under Study

In this section we describe three different multi-stage, tree-based topologies. In the 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 many 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.

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 denote the total number of switches in a topology as S, and the number of switches at level i as S_i. We call the relation between the number of downward ports of a switch and the number of upward ports the *over-subscription* or the *narrowing factor*. For example, look at the switches in the topologies shown in Fig. 1b and c, 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, so the over-subscription (or narrowing 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 denoting links 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,

$$\langle level, switch, port, \uparrow \rangle \Leftrightarrow \langle nlevel, nswitch, nport, \downarrow \rangle$$

In the graphical representations of the topologies in Fig. 1, boxes represent switches and lines represent links between them. Note that we do not show the compute nodes connected to the first level switches and their links as well as the last level upward links (which, as we stated before, are unplugged). These elements are hidden for the sake of simplicity. We also show some relations between S, S_i, N, n, k and k' for the topologies under study in Table 1.

	***k*,*n*-fat-tree**	***k*:*k'*,*n*-thin-tree**	***k*:*k'*,*n*-slender-tree**
Nodes	$N = k^n$	$N = k^n$	$N = k'^2 \cdot \left(\frac{k}{k'}\right)^n$
Switches	$S = n \cdot k^{n-1}$	$S = \sum_{i=0}^{n-1} k'^{(n-i)-1} \cdot k^i$	$S = \sum_{i=0}^{n-1} \frac{k^i}{k'^{i-1}}$
Switches per level: $\forall i \in [0, n-1]$	$S_i = k^{n-1}$	$S_i = k'^{(n-i)-1} \cdot k^i$	$S_i = \frac{k^i}{k'^{i-1}}$

Table 1. *Elements required to build the tree-like topologies.*

2.1 Fat-trees

The best-known of all the topologies considered in this study is the full fat-tree. This will be the yardstick against we will compare the other tree topologies.

Fat-trees are also called *k*-ary *n*-trees [9], whereby *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. In this paper we will denote them as *k,n*-fat-tree. Note that fat-trees do not have over-subscription; in other words, the narrowing factor is 1:1.

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

$$\forall level \in [0,\ n-2],\ \forall switch \in [0,\ k^{n-1}-1],\ \forall port \in [0,\ k-1]:$$
$$nlevel = level + 1$$
$$nswitch = \left(\left(port \cdot k^{level}\right) + \left(k^{level-1} \cdot \left\lfloor \frac{switch}{k^{level}+1} \right\rfloor\right) + \left(switch \bmod k^{level}\right)\right) \bmod k^{n-1}$$
$$nport = \left(\left\lfloor \frac{switch}{k^{level}} \right\rfloor - \left(k^{level+1} \cdot \left\lfloor \frac{switch}{k^{level}} \right\rfloor\right)\right) \bmod k$$

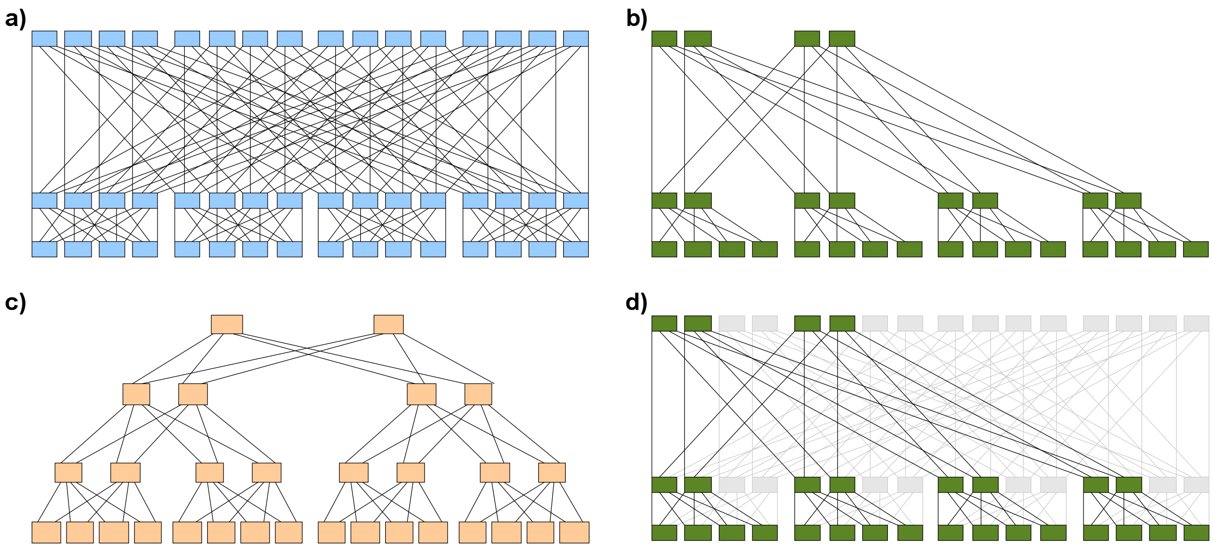

Fig. 1. *Samples of the 3 topologies under study to build a 64 nodes network. a) 4-ary 3-tree or 4,3-fat-tree. b) 4:2,3-thin-tree. c) 4:2,4-slender-tree. d) Superposition of the fat-tree and the thin-tree as a visual example of thin-tree construction*

The 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. However, 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 application of over-subscription: an attempt to reduce complexity in the upper levels without sacrificing application performance.

2.2 Thin-trees

We define a thin-tree as a cut-down version of a fat-tree in which we apply over-subscription. In a *k*:*k'*,*n*-thin-tree *k* is the number of downward ports, *k'* is the number of upward ports and *n* is the number of levels. The narrowing factor is, obviously, *k*:*k'*. In this topology, *k* does not need to be a multiple of *k'*. This means that we can produce a thin-tree with arbitrary values of *k* and *k'*. A *k*,*n*-fat-tree is actually a *k*:*k*,*n*-thin-tree.

A 4:2,3-thin-tree is depicted in Fig. 1b. In Fig. 1d we can see how a thin-tree can be seen as a fat-tree in which we have removed switches and links.

The topological neighborhood relationships between ports in a thin-tree can be described as follows:

$$\forall level \in [0,\ n-2],\ \forall switch \in \left[0,\ k\cdot\left\lfloor\frac{k}{k'}\right\rfloor^{n-level}-1\right],\ \forall port \in [0,\ k'-1]:$$

$$nlevel = level+1$$

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

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

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

2.3 Slender-trees

In both fat-trees and thin-trees, adaptive routing can be used to help packets advance in their upward path; however both topologies force a static downward path: once a lowest common ancestor between the source and destination is reached, the routing becomes oblivious because there is only one (shortest) path from this switch to the destination. As we are using local (not centralized) routing, it may occur that two packets going to the same destination coincide and are forced to share a link even if they are the only packets in the network at a given moment. When this occurs one message has to wait for the other—communication has to be serialized. Thus the delay for these messages can be much longer than expected in an almost empty network.

We present in this paper the slender-tree topology, which tries to reduce the above-mentioned inefficiency. A slender tree is also a narrowed tree, where cost and complexity are reduced by reducing the number of switches per level. It is *similar* to the thin-tree, but not exactly equal. It provides path diversity in the upward part of the message path from the source node (as in the case of fat and thin-trees), and *also* in the downward part.

In a *k*:*k'*,*n*-slender-tree *k* is the number of downward ports, *k'* is the number of upward ports and *n* is the number of levels. For this topology, *k* and *k'* must fulfill some restrictions: *k'* has to be lesser than *k*, and *k* has to be divisible by *k'*. Fig. 1c shows a depiction of a 4:2,4-slender-tree topology. The topological neighborhood relationships between ports in a slender-tree are expressed as follows:

$$\forall level \in [0,\ n-2],\ \forall switch \in \left[0,\ k'\left\lfloor \frac{k}{k'} \right\rfloor^{n-1-level} -1\right],\ \forall port \in [0,\ k'-1]:$$

$$nlevel = level + 1$$

$$nswitch = k' \cdot \left\lfloor \frac{switch}{k} \right\rfloor + port$$

$$nport = switch \bmod k$$

As we just stated, the slender-tree allows for adaptivity in the downward paths. However, it has two clear weak points. The first one is that it has a low bisection bandwidth, which is even lower than that of an equivalent thin-tree; thus, this topology does not work well under heavy loads that have to travel through the higher levels of the network. The second weakness is that, to create a network with a given number of leaves, slender-trees need more levels than fat-trees or thin-trees— meaning that the average path length is higher.

Note that the height (number of levels) of slender-trees is derived from the way they scale. A *k*:*k’*,(*n*+1)-slender-tree can be built with *k’* smaller *k*:*k’*,*n*-slender-trees. In contrast, a *k*:*k’*,(*n*+1)-thin-tree contains *k* smaller *k*:*k’*,*n*-thin-trees (same for fat-trees). As usually *k’*<*k*, slender-trees are easier to adjust to a specific network size, an interesting property when we want to upgrade an existing machine. In Fig. 1 we see how we have to jump from a 4:2,2-thin-tree (16 nodes) to a 4:2,3-thin-tree (64 nodes). In the slender-tree case, we can go from the 16-node network (4:2,2-slender-tree, identical to a 4:2,2-thin-tree) to a 32-node one (4:2,3-slender-tree).

3 Experimental Set-up

We use simulation [10] to test some different tree-like topologies, feeding them with a collection of traffic patterns. The simulator measures time in terms of *cycles*. A cycle is the time required by a phit to traverse one switch.

3.1 Switches

For this work, we have chosen simple input-buffered switches whose radixes range from 10 to 16, depending on the topology. In order to keep things simple, we do not use virtual channels. The arbitration of each output port is performed in a random way, that is, every time an output port is free it randomly chooses 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. There is a schematic model of the switch in Fig. 2.

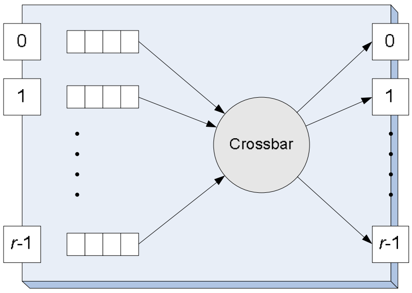

Fig. 2. *Model of the switch used in the simulations given a radix* r.

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 to 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 applications traffic.

Messages are split into packets of a fixed size of 16 phits. One phit is the smallest transmission unit, fixed to 32 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, when possible, adaptive using shortest paths. A credit-based flow-control mechanism is used, so that when several output ports are possible, the one with more available credits is selected. Credits are communicated out-of-band, so they do not interfere with normal traffic.

3.2 Networks under study

In this work we have performed two sets of evaluations for the three topologies under study. As we stated before, slender-trees have to be built obeying some restrictions. As we want to compare them with thin-trees in a fair way, we will use the same levels of narrowing factor for both topologies.

In the first evaluation set we have fixed the number of downward ports per switch (set to 8) and over-subscription rates (set to 2:1 and 4:1). We have also fixed the target number of connectable compute nodes to 4096. With these restrictions, we have worked with the following topologies: 8,4-fat-tree, 8:4,4-thin-tree, 8:2,4-thin-tree, 8:4,8 slender-tree and 8:2,5-slender-tree.

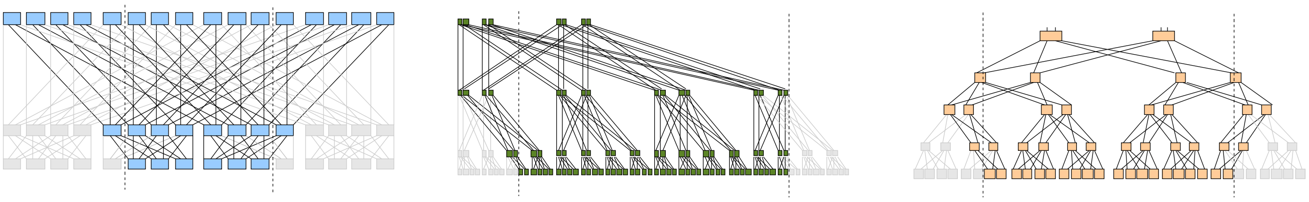

Fig. 3. *"Partial" construction of the topologies: 4,3-fat-tree (left), 4:2,4-thin-tree (middle) and 4:2,5-slender-tree (right).*

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 “narrowed” topologies. Thus, in this first evaluation the fat-tree has advantage compared with the narrowed alternatives: it uses more links, and more switches that also are larger. Raw performance measurements are biased towards the fat-tree topology.

In the second evaluation set we have fixed the radix of the switches (set to 12), and used the same over-subscription rates for thin and slender-trees, in this case 2:1 and 3:1. Under these restrictions we have created the smallest topologies capable to connect at least 4096 nodes: 6,5-fat-tree, 8:4,4-thin-tree, 9:3,4-thin-tree, 8:4,8-slender-tree and 9:3,6-slender-tree. 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 connectivity than fat-trees, but locality is increased.

Unfortunately, studied networks have different sizes, in terms of the number of compute nodes they can connect (*N*). As we are using workloads that emulate a fixed number of tasks (4096), we are not capable of using all the trees' leaves. This means that results will be biased against the smallest networks in number of nodes (2:1 narrowed slender and thin-trees). In order to reduce this bias, we have placed the tasks in the central nodes of the networks, maximizing the use of the links in the higher levels. By doing this, there are two symmetrical parts of the systems that are not actually used. Fig. 3 shows examples of partially used topologies. Vertical dotted lines delimit the part of the system that is in use (the nodes with active tasks) and shaded switches and links represent unused elements. Note how unused switches are always located in the lower levels, so the upper levels are complete.

a)	8,4-fat-tree	8:4,4-thin-tree	8:2,4-thin-tree	8:4,8-slender-tree	8:2,5-slender-tree
Switch radix	16	12	10	12	10
Switches	2048	960	680	1020	682
Links	16384	7680	5440	8160	5456
Nodes	4096	4096	4096	4096	4096

b)	6,5-fat-tree	8:4,4-thin-tree	9:3,4-thin-tree	8:4,8-slender-tree	9:3,6-slender-tree
Switch radix	12	12	12	12	12
Switches-CN	6480	960	1080	1020	1092
Links-CN	38880	7680	9720	8160	9828
Nodes-CN	7776	4096	6561	4096	6561
Switches-U	4248	960	700	1020	694
Links-U	21808	7680	6115	8160	6169
Nodes-U	4096	4096	4096	4096	4096

Table 2. *Characteristics of the topologies under study. a) For the experiments with topologies with same size but different switch radix. b) For the experiments with radix 12 switches, considering the complete network* (CN) *and only the actually used* (U) *resources.*

Table 2 summarizes some characteristics of the studied networks. Note how it is possible to build 8:4,4-thin and slender-trees with exactly 4096 nodes, but the complete 6,5-fat-tree built with 12-port switches has 7776 leave nodes. In terms of cost this plays against the topologies that provide the worst fit to the target number of nodes.

3.3 Workloads

We would 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, so we decided to create instead some (synthetic) traffic generators which emulate data interchanges typically used in parallel programs.

The patterns of choice have been: wave-front movements in 2D and 3D virtual meshes, nearest-neighbor interchanges in 2D and 3D virtual meshes, binary tree interchanges, butterfly interchanges and a traffic pattern that we have denominated *waterfall*. A graphical representation of these patterns is depicted in Fig. 4. In the descriptions that follow, *N* is the number of processes in the parallel application. Note that we assume a mapping of one process to one compute node attached to one leave of the network.

For all patterns, except for waterfall, each node starts with a set of *messages* (it can be an empty set) that have to be injected into the network. The size 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 an additional one, of the same size—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. The way messages are generated in the waterfall pattern will be discussed when describing this pattern.

We have performed simulations with message sizes 10, 100, 1 K, 10 K and 100 K bytes, for a fixed network size of *N*=4096 compute nodes—this means using a complete network of exactly 4096 nodes or, as explained before, the 4096 central nodes of a larger network. In the following paragraphs we provide a definition of these patterns. In the descriptions, we use *N* communicating processes, identified from 0 to *N*-1, and a flat network, so that when talking about *distance* between two nodes we mean difference between identifiers.

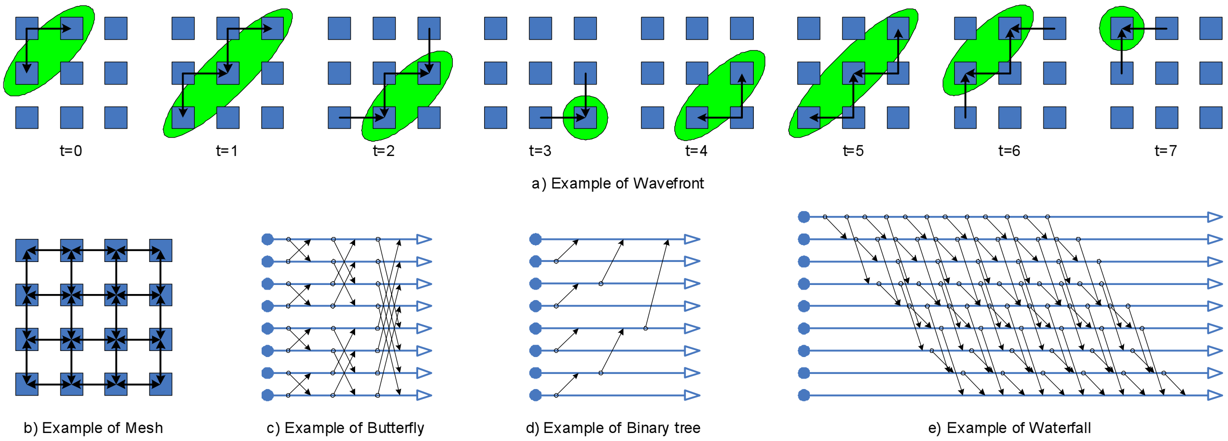

Fig. 4. *Graphical representation of the used traffic patterns: Time flows from left to right. Blue lines and squares represent nodes. Each black arrow start means a message send. The end of the black arrows means that the node has to stop until receiving the corresponding message. a) Wave-front (Green)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

The 2D and 3D wave-front patterns (**2W** and **3W**) 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

3W) messages in node 0, and ends with the finalization of the (return) sweep. These patterns do not impose a very heavy load on the network – 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. Regarding message distance, in 2W it can take just two values: 1 and $\sqrt{N}$. In the case of 3W, it can take three values: 1, $\sqrt[3]{N}$ and $\sqrt[3]{N^2}$. We can observe this pattern in applications implementing the Symmetric Successive Over-Relaxation (SSOR) [4] algorithm—used to solve sparse, triangular linear systems.

The 2D and 3D mesh patterns (**2M**, **3M**) 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 2M, 3-6 for 3M), and ends when all the response messages have arrived. 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. Regarding message distance, in 2M it can take just four values: 1 and $\sqrt{N}$. In the case of 3M, it can take values: 1, $\sqrt[3]{N}$ and $\sqrt[3]{N^2}$. These patterns can be observed in applications using finite difference methods [1].

The butterfly pattern (**BU**) provides an efficient implementation of MPI N-to-N collectives (MPI_Alltoall, MPI_Allreduce, etc.) [11]. It is also known as "recursive doubling", or "recursive halving" in the inverse butterfly case. BU is O(N log N) in number of messages and O(log N) in time. Alternative implementations of these collectives could be a 2-round ring (O(N) messages; O(N) time) or a burst of point-to-point operations from each node to the rest (O(N^2) messages; O(N) time). Simulation of BU starts with a message at each node, and ends when defined by the pattern. Message distances range from 1 to (N/2).

The binary tree pattern (**BT**) provides an efficient implementation of some N-to-1 MPI collective operations, such as MPI_Reduce and MPI_Gather [7]. In a similar way, 1-to-N collectives (MPI_Broadcast and MPI_Scatter) are commonly implemented using an inverse binary tree. A usual mechanism to implement MPI_Barrier is by concatenating a binary tree and an inverse binary tree. In the BT pattern, message interchanges are performed in O(N) in number of messages and O(log N) in time. Alternative implementations, in which every node sends a message to a "root" node, the number of messages is again O(N), but the time grows to O(N). Simulation of this pattern starts with a message at odd-numbered nodes, and ends when node 0 receives the last message from the all power of two identified nodes (included 2^0=1). Message distances range from 1 to (N/2).

The waterfall traffic pattern (**WF**) is inspired in a pattern we have observed in the NAS Parallel Benchmark LU. It consists of a large collection of small messages, with causal dependencies, sent to distances 1 and $\sqrt{N}$. For this pattern, we define a total number of bytes to transmit (*size*) and, instead of starting with a single, large message, a collection of small messages is generated. We have modeled the length of these messages (*mlength*) as $8 \cdot 2^{\log_{10} size}$ Bytes. The spatial pattern of WF is the same of the first half of 2W (actually, LU implements SSOR); however, the causal relationships are different. They are described in Fig. 5. The simulation starts with the collection of messages at node 0, and ends when the last message, as defined by the pattern, is consumed in node *N*-1. The main characteristic of WF is the length of the causality chains: in our 4096-node network, this length is 64. Latencies in the delivery of messages accumulate at the end of the chain.

```
$\forall node \in [0,\ N-1]$
repeat $\left\lceil \frac{size}{mlength} \right\rceil$ times:
    if $node \bmod \sqrt{N} \neq 0$
        wait_message_from( $node - 1$ )
    if $node \geq \sqrt{N}$
        wait_message_from( $node - \sqrt{N}$ )
    if $node \bmod \sqrt{N} \neq \sqrt{N} - 1$
        send_message_to( $node + 1$ )
    if $node \leq N - \sqrt{N}$
        send_message_to( $node + \sqrt{N}$ )
```

Fig. 5. *Pseudo-code that represents the causal relationships of the WF pattern.*

4 Experiments and Analysis of Results

4.1 Experiments with same size networks

In the first set of experiments we have gathered the time (in simulation cycles) required to deliver all the messages for each different traffic pattern in the networks under study. As these times differ widely, due to the characteristics of the patterns and to the different message sizes, we have normalized them, using the times for the fat-trees as the reference. These normalized times are represented in Fig. 6.

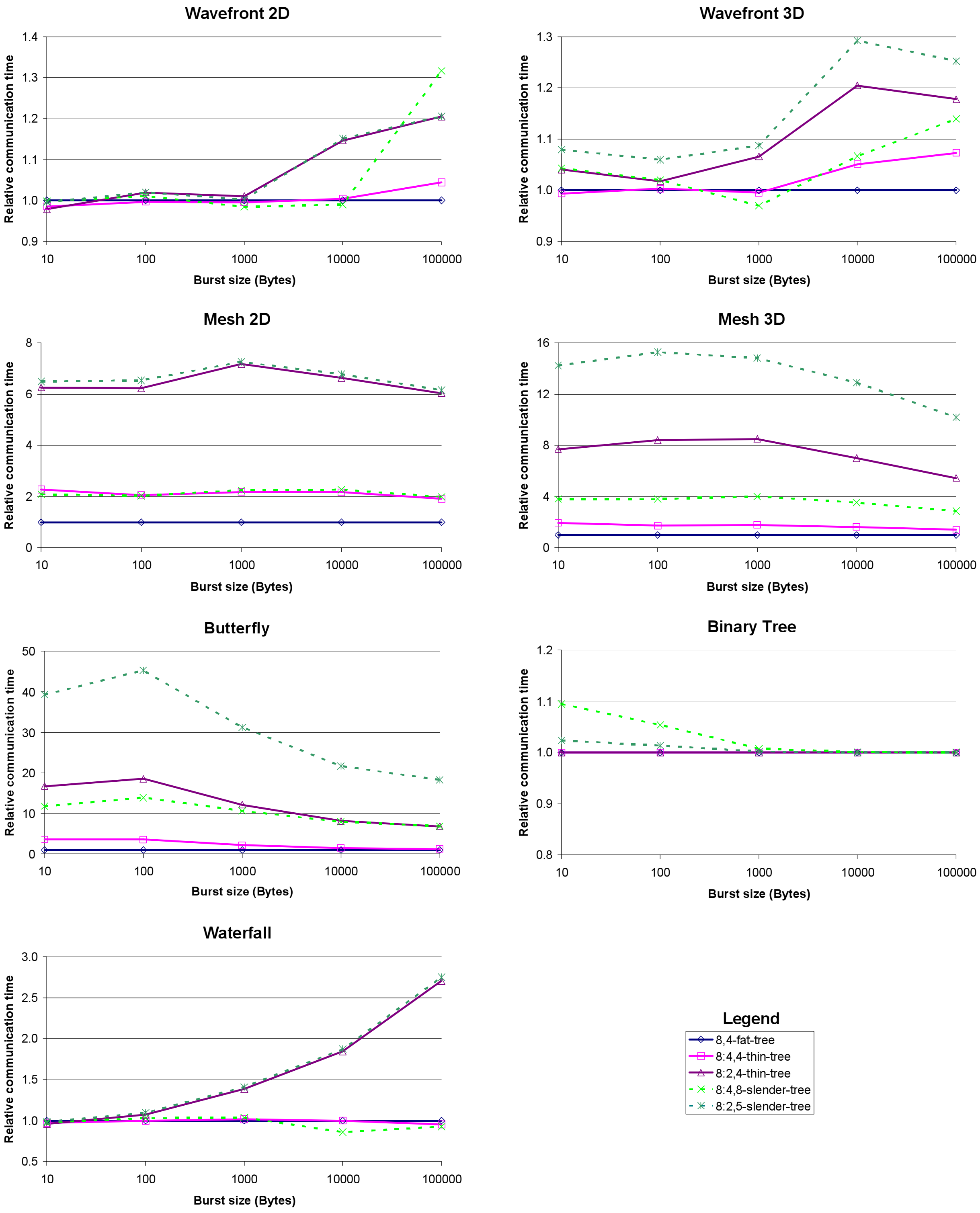

Fig. 6. *Normalized time to perform all communications of each traffic pattern in the same-sized networks.*

We can observe that both thin-tree and slender-tree topologies perform acceptably well in “light” patterns, using this term to describe those patterns in which the number of messages circulating simultaneously through the network is low, and messages are short. We consider that patterns like 3W, 2W, WF and BT are light. With them, the fat-tree cannot take advantage of its high bandwidth and path diversity. Note, though, the bad behavior of the slender-tree for the BT pattern with small messages: as there is only one short message (meaning just a few packets) in each zone of the network at once, slender-trees are penalized by their longer distance; adaptivity is useless because there are too few packets simultaneously in the network.

Note that in all the experiments in this set switches have the same number of downward ports (8); thus, the performance of the thin-tree is delimited by that of the fat-tree and cannot be better. However, in some cases, the communication time for the slender-tree is smaller than that of the fat-tree, due to the possibility of adaptation in both the upward and the downward paths. Regarding the over-subscription factor in the topologies, the 2:1 narrowed topologies perform notably better than the ones with 4:1 in almost all the cases. This is because upward resources in the narrowest topologies do not provide enough bandwidth to achieve the performance of the “not-so-narrowed” ones.

We can conclude from this set of experiments that, if we ignore costs, the fat-tree is the best-performing topology in almost all experiments (combinations of pattern and message size). However, in many cases narrowed topologies with 1:2 over-subscription perform equally well—in some cases, even better. Additional narrowing causes excessive network contention, so results are noticeably worst. Moreover, in almost all cases, thin-trees are faster than slender-trees.

4.2 Experiments with same radix switches

In this section we analyze the results of the experiments comparing topologies built with switches with the same size (12 ports). Remember that we build topologies for 4096 nodes *or more*. In all cases we use only the 4096 central nodes, meaning that there may be many unused links and switch ports.

Results are summarized in Fig. 7. In general, they confirm what we learned from the previous set of experiments. However, in this case, the narrowed topologies have an additional advantage: the increased ability to exploit locality in communication. A switch in a 6,5-fat-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 narrowed 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 narrowed topologies beat the performance of fat-trees in many cases. For example, the BU traffic with long messages is one of the heaviest workloads, and the 8:4,4-thin-tree is able to deliver it in less time than the fat-tree. In most of the cases the thin-trees perform better than the comparable slender-trees.

The WF pattern requires specific attention. Note the excellent performance of narrowed trees. The high causality of this pattern does not allow the utilization of all the resources of the fat-tree, but allows for a productive exploitation of additional levels of locality. Again, over-subscription should not go very far. The performance of the narrowest topologies (9:3,4-thin-tree and 9:3,5-slender-tree) is too low for the heavier workloads. Still, their performance is good for the lighter workloads.

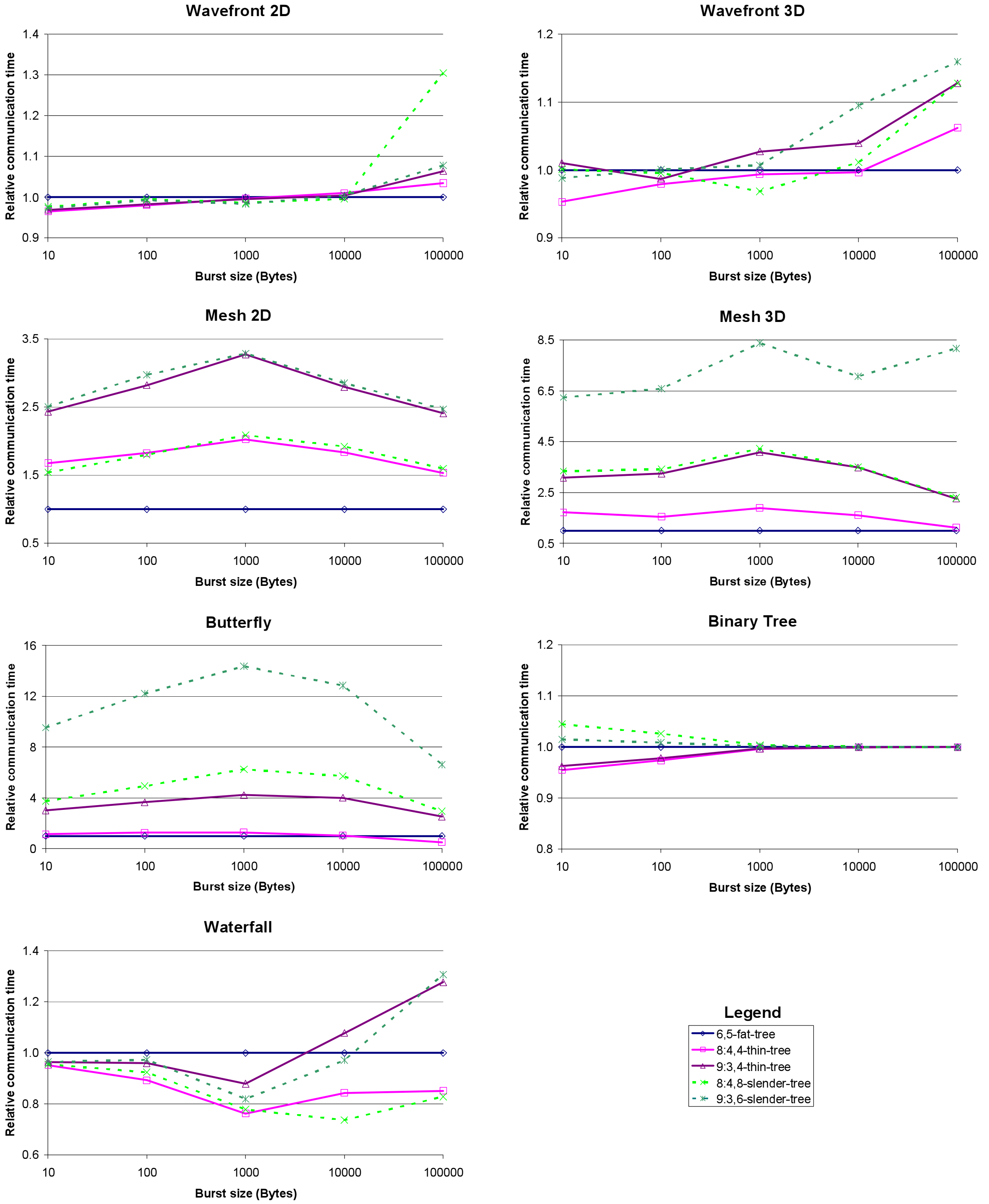

Fig. 7. *Normalized time to perform all communications of each traffic pattern in the networks with radix-12 switches.*

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, narrowed topologies with 1:2 over-subscription, thin-trees in particular, perform as well as, or even better, than fat-trees. Going to higher levels of over-subscription is counterproductive.

5 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 will propose a cost function for each topology, in absolute terms and in cost per attached node. Furthermore, it is necessary to measure the performance of the network by evaluating it with appropriate workloads. We will propose a simple way to characterize the performance of the network. Also we show some examples of application workloads characterization.

5.1 Cost of the networks

The characteristics of the networks under evaluation, gathered in Table 2, provide us with clues about their cost. If C_l is the cost of a link, and C_s is the cost of a switch, the total cost would be

$$Cost = C_l \cdot num\_links + C_s \cdot num\_switches$$

We need the values of C_l and C_s. Unfortunately, these values are not constants. C_l depends on the length of the link, and C_s on the complexity of the switches—number of ports. These dependencies are not necessarily linear.

We have checked the prices of actual equipment manufactured by Myricom: the Myrinet-2000 range of products, available at its web site [8]. The prices of links (fibre pairs) range from \$70 for the 1 m. link to \$350 for the 200 m. link. For simplicity, we will assign a constant value of \$150 to C_l (this is the cost of the 25 m. link). This way we do not take into account the differences in link lengths at the different levels

For the switches, we take as a reference the cost of a 16-port switch: \$5625. We assume that the cost of a switch increases quadratically with the number of ports (technologically this is true, although in real systems a 12-port switch might be a 16-port switch with different packaging). This allows us to estimate the cost of smaller (or larger) switches.

$$C_s(nports) = \frac{5625}{16^2} nports^2 = 21.97 nports^2$$

For the cases under study, $C_s(10)$ = \$2199, and $C_s(12)$ = \$3167.

The cost of the network should include, too, the price of the adapters attached to the compute nodes and the links from the adapters to the first switch. This cost is independent of the topology of choice, so we have assumed that it simply increases the price of the compute nodes, and do not include it in our computations.

We have gathered, in Table 3, the total cost of the topologies under study, and the cost per attached node. For those topologies with more than 4096 nodes, we have calculated the costs for the complete network (CN) and also for the partial networks in which only the 4096 central nodes are used (U).

Note how 2:1 over-subscription reduces price to less than 1/3. Higher levels of over-subscription do not generate savings of the same degree. The cost of comparable narrowed trees is very similar, although thin-trees are slightly cheaper than slender-trees.

a)	8,4-fat-tree	8:4,4-thin-tree	8:2,4-thin-tree	8:4,8-slender-tree	8:2,5-slender-tree
Total Cost	13977600	4189500	2310141	4451344	2316935
Cost/Node	3413	1023	564	1087	566

b)	6,5-fat-tree	8:4,4-thin-tree	9:3,4-thin-tree	8:4,8-slender-tree	9:3,6-slender-tree
Total Cost-CN	26335125	4189500	4875188	4451344	4929356
Cost/Node-CN	3387	1023	743	1087	751
Total Cost-U	16712138	4189500	3132094	4451344	3121209
Cost/Node-U	4080	1023	765	1087	762

Table 3. *Total costs, and cost per node, for the studied topologies. a) For the experiments with topologies with same size but different switch radix. b) For the experiments with switches with radix 12, considering the complete network* (CN)*, and only the 4096 central nodes and the actually used* (U) *resources.*

5.2 Characterizing performance

If we had unlimited (financial) resources we could just select the best-performing option, but that option may (or may not) be the most cost-effective. As we know how to measure the cost, what we need now is a means to measure the effectiveness of a network, which depends on 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, but that can be extended with further workload types.

For each given traffic pattern (*P*) and burst size (*S*) the simulation reports a (relative) time $T_{P,S}$. For example, we have a certain (relative) time $T_{2M,1000}$ for the experiment with the 2-D Mesh pattern, for messages of 1Kbytes, and a different $T_{BU,100000}$ for Butterfly pattern with messages of 100 Kbytes. Note that these values are relative, so they are always 1 for the fat-trees.

Depending of the application mix of interest in a particular computing center, we may apply a weighting factor to each experiment. For example, this weight could be 0 for all applications with small messages, if they are never used. As we have 7 patterns and 5 burst sizes, we would need to use 35 different weights. Alternatively, we can use a weight per pattern (w_P) that indicates how important a given application is in our set, and define also a weight per burst size (w_S) that indicates the relevance of a given range of sizes. This means using 12 different values, instead of 35. Thus, for a given network, its performance $\boldsymbol{\varphi}$ would be expressed as

$$\varphi = 1 \Big/ w_{2M} w_{10} T_{2M,10} + w_{2M} w_{100} T_{2M,100} + \ldots + w_{2M} w_{100000} T_{2M,10000} + w_{3M} w_{10} T_{3M,10} + \ldots + w_{WF} w_{100000} T_{WF,100000}$$

For a given application mix (set of weights) a high value of ***φ*** represents a well-performing topology.

As we cannot identify all "representative" application mixes, we have decided to use 1 for all the weights in order to provide an example of the proposed methodology. With this, the denominator in our efficiency value is just the addition of the (relative) times obtained in the experiments. This yields a value of ***φ***=1/35 for the fat-tree. We further normalize this value multiplying it by 35. Table 4 shows the normalized performance values for all the experiments. Note how, using this criterion, the best performing network for the first set of experiments is the 8,4-fat-tree. For the second set, we can see that 6,5-fat-trees and 8:4,4-thin trees are the best performers, with almost identical ***φ***.

5.3 Performance/cost

Once we have characterized cost and performance, the performance/cost ratio can easily be computed. The higher this ratio, the better: it means that we are getting more value from our money. The ratios for the considered topologies are included in Table 4.

a)	**8,4-fat-tree**	**8:4,4-thin-tree**	**8:2,4-thin-tree**	**8:4,8-slender-tree**	**8:2,5-slender-tree**
∑ Relative time	35.00	46.95	131.66	91.17	243.85
Normalized *φ*	1.00	0.75	0.27	0.38	0.14
Cost/Node	3413	1023	564	1087	566
Norm Cost/Node	1.00	0.30	0.17	0.32	0.17
Perf./Cost	1.00	2.49	1.61	1.21	0.87

b)	**6,5-fat-tree**	**8:4,4-thin-tree**	**9:3,4-thin-tree**	**8:4,8-slender-tree**	**9:3,6-slender-tree**
∑ Relative time	35.00	34.17	55.74	55.77	100.10
Normalized *φ*	1.00	1.02	0.63	0.63	0.35
Cost/Node-CN	3387	1023	743	1087	751
Norm Cost/Node-CN	1.00	0.30	0.22	0.32	0.22
Perf./Cost-CN	1.00	3.39	2.86	1.96	1.58
Cost/Node-U	4080	1023	765	1087	762
Norm Cost/Node-U	1.00	0.25	0.19	0.27	0.19
Perf./Cost-U	1.00	4.09	3.35	2.36	1.87

Table 4. *Performance/Cost analysis results using the same weight for all the experiments. a) For same size networks. b) For same radix networks, taking into account the complete network* (CN) *and only the used elements* (U).

Note the excellent characteristics of the 8:4,4-thin-tree: its raw performance is the same of the full tree (6,5-fat-tree), and the cost is four times smaller. The remaining narrowed topologies also have better performance/cost ratios than the fat-tree, although they win only because of the lower cost.

To further explore this analysis, we consider two different example scenarios. In the first one, we want to build a system composed by exactly 4096 nodes, and we can choose switches of different radix. Target applications perform mostly MPI collective operations. In this case, we explore the performance/cost ratio of our first set of experiments, focusing on BU and BT patterns—to which we assign a weight of 1. The remaining patterns receive weight 0. Regarding the weights for the message sizes, we set all of them to 1. We can see in Table 5a how the fat-tree is clearly the best option in terms of raw performance. However, when cost is also considered, the 2:1 narrowed thin-tree is still the most cost-effective option.

The second example scenario is a production machine built with off-the-shelf switches, all of them with radix 12. We will use it to run several scientific applications whose only common characteristic is that all of them work with large problems that always generate long messages. For this application mix we only consider workloads with the longest burst size (100 Kbytes). In other words w_{10}=w_{100}=w_{1000}=w_{10000}=0 and w_{100000}=1. The weight for all traffic patterns will be 1. In this case, we take into account the second set of experiments and the cost per node for the used elements only. The performance indicator for the fat-tree is φ=1/7. Normalizing it to 1, the performance of all the topologies, and the performance/cost ratio is in Table 5b. We can see that, in this case, the preferred system in terms of raw performance is the 2:1 narrowed thin-tree. This winning position is even clearer when considering the performance/cost ratio.

a)	8,4-fat-tree	8:4,4-thin-tree	8:2,4-thin-tree	8:4,8-slender-tree	8:2,5-slender-tree
∑ Weigth time	10.00	17.00	67.43	56.29	160.82
Normalized φ	1.00	0.59	0.15	0.18	0.06
Cost/Node	3413	1023	564	1087	566
Norm Cost/Node	1.00	0.30	0.17	0.32	0.17
Perf./Cost	1.00	1.96	0.90	0.56	0.38

b)	6,5-fat-tree	8:4,4-thin-tree	9:3,4-thin-tree	8:4,8-slender-tree	9:3,6-slender-tree
∑ Weigth time	7.00	6.39	10.58	9.47	15.19
Normalized φ	1.00	1.09	0.66	0.74	0.46
Cost/Node-U	4080	1023	765	1087	762
Norm Cost/Node-U	1.00	0.25	0.19	0.27	0.19
Perf./Cost-U	1.00	4.37	3.53	2.77	2.47

Table 5. *Performance/Cost analysis for the suggested examples. a) System to run collective-based applications. b) Scientific production machine.*

We want to remark that the performance/cost analysis carried out in this section is exceedingly simplistic, for several reasons—some of them obvious, some others less evident.

Regarding the performance indicator, notice that we have used only network times to compute it, that is, the time used by a given network to consume a given workload. Actual applications running on actual machines include a mixture of network, CPU and I/O usage. 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 a 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 is application-dependent.

Furthermore, the collection of workloads used in this work might not be representative of actual workloads used at supercomputing centers. A thorough study should be customized for a particular site, taking into account their applications and relative weights.

Another important point is the exploitation of locality in our evaluations: locality gives advantage to narrowed topologies.

In all of our experiments we have assumed that we use the whole network to run a single application and have arranged the nodes in an increasing order. Most supercomputing centers cannot make this assumption. They use a supercomputer (cluster, MPP) to run many applications simultaneously, often via space-sharing of nodes. Typical schedulers for these machines do not include locality in their resource assignment algorithms, so that the collection of compute nodes on which a parallel application runs can be spread across the entire network, using non-contiguous resources; in other words, they assume *flat* networks. A notable exception—but by no means the only one—is the scheduler for the BlueGene/L, which assigns jobs to rectangular-shaped blocks of contiguous nodes [2]. Under the flat network and multiple jobs assumptions, the locality advantage disappears; new evaluations, with different workloads, would be required to assign performance indicators to the topologies under study. Although that would go beyond the scope of this paper, it is intuitively clear that higher network delays must be expected, but it remains to be seen to which degree the higher network bandwidth of the fat-tree in the upper levels changes the relative difference with the narrowed fat-tree topologies.

6 Conclusions and future work

We have presented and characterized two alternative, narrowed versions of fat-trees: *thin-trees* and *slender-trees*. A thin-tree can be seen as a fat-tree after removing some links and switches; a slender-tree is slightly different, and has the theoretical advantage of making adaptivity possible in the downlinks—the price to pay is a network with more levels. A narrowed tree costs a fraction of the price of a fat-tree (less than 1/3, for the networks under study). Cost of similar thin and slender-trees are approximately the same. In terms of performance, thin-trees with low levels of over-subscription perform as well as comparable fat-trees. Excessive over-subscription (beyond 2:1) results in bad performance results. A comparison of performance/cost ratios turns out to be very favorable for the narrowed trees in all the studied cases.

Narrowed trees are obviously cheaper than the fat-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 narrowed tree devotes more ports to downward links, so more nodes can communicate without using the upper levels.

Two supercomputers that currently are in positions #6 (Thunderbird) and #9 (Tsubame) of the TOP500 list [5] have been built with Infiniband networks arranged as narrowed trees—actually, spines. In Thunderbird the over-subscription is 2:1; Tsubame goes further, to 5:1. 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 fat-trees, narrowed topologies provide comparable performances at *much* lower cost.

This study opens several lines for future work. We want to further explore the workload generation mechanism, to make it mimic the characteristics of supercomputing applications as accurately as possible. One interesting place to start is [3], 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 narrowed 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/L [6], we need to make scheduler’s topology aware.

Acknowledges

This work has been done with the support of the Spanish Ministerio de Educación y Ciencia, grants TIN2004-07440-C02-02 and TIN2007-68023-C02-02. Mr. J. Navaridas is supported by a doctoral grant of the UPV/EHU.

We gratefully acknowledge the support of the Barcelona Supercomputing Center / Centro Nacional de Supercomputación in giving shape to the ideas discussed in this paper.

References

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

[2] 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

[3] 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

[4] 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.

[5] J.J. Dongarra, H.W. Meuer, E. Strohmaier. “Top500 Supercomputer sites”. November of 2006 edition. Available at: http://www.top500.org/

[6] E. Krevat, J. Castaños, and J. Moreira. “Job scheduling for the Blue Gene/L system”. In Job Scheduling Strategies for Parallel Processing, volume 2537 of Lecture Notes in Computer Science, pages 38–54. Springer, 2002.

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

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

[9] 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, pages 87–93, Geneva, Switzerland, April 1997.

[10] 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).

[11] 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