Peñaranda Cebrián, R.; Gómez Requena, C.; Gómez Requena, ME.; López Rodríguez, PJ.; Duato Marín, JF. (2016). The k-ary n-direct s-indirect family of topologies for large-scale interconnection networks. Journal of Supercomputing. 72(3):1035-1062. doi:10.1007/s11227-016-1640-z.

Noname manuscript No.
(will be inserted by the editor)

The k-ary n-direct s-indirect Family of Topologies for Large-Scale Interconnection Networks

Roberto Peñaranda · Crispín Gómez · María E. Gómez · Pedro López · Jose Duato

Received: date / Accepted: date

Abstract In large-scale supercomputers, the interconnection network plays a key role in system performance. Network topology highly defines the performance and cost of the interconnection network. Direct topologies are sometimes used due to its reduced hardware cost, but the number of network dimensions is limited by the physical 3-D space, which leads to an increase of the communication latency and a reduction of network throughput for large machines. Indirect topologies can provide better performance for large machines, but at higher hardware cost. In this paper, we propose a new family of hybrid topologies, the k–ary n–direct s–indirect, that combines the best features from both direct and indirect topologies to efficiently connect an extremely high number of processing nodes. The proposed network is an n–dimensional topology where the k nodes of each dimension are connected through a small indirect topology of s stages. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to indirect topologies, but at a lower hardware cost. In particular, it doubles the throughput obtained per cost unit compared with indirect topologies in most of the cases. Moreover, their fault-tolerance degree is similar to the one achieved by direct topologies built with switches with the same number of ports.

Keywords High-Performance Computing · Interconnection Networks · Direct Topologies · Indirect Topologies · Hybrid Topologies · Routing

1 Introduction

The size of large supercomputers has been growing year after year. The topmost machines of the top 500 supercomputer list [6] are being built up by using

Roberto Peñaranda
Universidad Politécnica de Valencia
E-mail: ropeaceb@gap.upv.es

hundreds of thousands of processing nodes. All these processing nodes work jointly to solve a given problem in as less time as possible. This joint work is performed thanks to the interconnection network that allows all the processing nodes to share data among them. The interconnection network must enable an efficient communication among all the processing nodes because it strongly impacts the performance of the whole system. Transmission time of data across the interconnection network adds up to the processing time, impacting the time required to run the applications.

The main design challenges of interconnection networks design are to provide low latency communications, in order to reduce the execution time of applications, and a high network throughput, to allow as many simultaneous communications as needed, while providing a simple implementation at a reduced hardware cost.

In addition to performance and cost, another important feature of interconnection networks is their ability to provide fault–tolerance. The high amount of hardware that can be found in an interconnection network in high–performance machines significantly impacts the probability of having a fault in the system. Each component may independently fail, and therefore, the probability of having a single fault in the whole system drastically raises with the number of elements that compose it. Thus, tolerating faults is also a basic requirement when designing an interconnection network, specially for a large machine.

Two of the most important design issues of the interconnection networks are the topology and the routing algorithm [11,15]. Topology defines how the components of the system are connected, and the routing algorithm determines the path that is followed by packets from their source to their destination node. The topology of a network also impacts, to a large extent, its cost. Topologies usually adopt a regular structure to simplify their implementation, the routing algorithm and the possibility of expanding the network. Among the different taxonomies of regular topologies, the most commonly-used one divides them into direct and indirect topologies [11,15].

Direct topologies usually adopt an orthogonal structure where nodes are organized in an n–dimensional space. Each node is composed of a router and a processing node. The nodes are connected in each dimension in a ring or array fashion. 2D or 3D direct topologies are relatively easy to built as each topology dimension is mapped to a physical dimension. Implementing direct topologies with more than three dimensions implies not only increasing its wiring complexity but also the length of its links when they are mapped to the 3D physical space. Indeed, the number of ports of the routers geometrically grows with the number of dimensions (as two ports per each dimension are required). The implementation limitation in the number of dimensions leads to increase the number of nodes per dimension, which increases the communication latency, negatively impacting performance. As a consequence, direct topologies are not the most suitable ones for large machines.

The alternative is to use an indirect topology. The main difference, compared to direct topologies, is that not all the routers have an associated processing node. The most common indirect topologies are multistage indirect

networks (MINs) where switches are organized in a set of n stages. Indirect topologies provide better performance for a large number of processing nodes than direct ones. However, this is achieved by using a higher amount of switches and links. Furthermore, their physical implementation is complex due to the fact that the wiring complexity grows with the number of processing nodes in the system, unlike direct topologies where complexity grows with the number of topology dimensions.

To overcome the limitations of direct and indirect topologies, in this paper we propose a new family of topologies where we combine the best features of both types of topologies. Other hybrid topologies have been proposed previously in the literature (see Section 6). However, most of them are hierarchical and therefore introduce long latency for the communication of non-local processing nodes and others are intended for particular purposes [20,10]. In this paper we propose a topology family that can be configured to meet different scenario requirements. We propose an n–dimensional topology, where the rings that connects the nodes in each dimension are replaced by small indirect networks. In this way, communication latency along each dimension no longer linearly grows with the number of nodes per dimension thanks to these indirect networks. On the other hand, the small size of this indirect topology allows a reasonable wiring complexity opposite to large indirect topologies. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but at a reduced hardware cost. In addition, the fault–tolerance level is higher than or equal to the one provided by direct and indirect topologies.

Evaluation results show that the new proposed family of topologies can obtain similar or better performance results than the ones provided by indirect topologies, but using a smaller amount of hardware resources and with an easier implementation. In particular, they are able to double the throughput obtained per cost unit compared to the one obtained with indirect topologies in most of cases, and this difference is even higher when it is compared with direct topologies.

The rest of the paper is organized as follows. Section 2 presents some background. Section 3 describes the proposed family of topologies. The routing algorithms to be used are presented in Section 4. Section 5 evaluates the new family of topologies, comparing it against direct and indirect topologies, and other recently proposed topologies. Some related works are commented in Section 6. Finally, some conclusions are drawn.

2 Direct and Indirect Topologies

In direct topologies, each node has its own router that connects it to a subset of the nodes of the system by means of point-to-point links. Direct topologies usually adopt an orthogonal structure, where nodes are organized in an n–dimensional space, in such a way that traversing one link produces a dis-

placement in only one dimension. That is, all the links of the network are organized in several dimensions in a regular way, and each node has at least one link in each dimension. The symmetry and regularity of these networks greatly simplify its implementation and the routing algorithm, since the movement of a packet in a single dimension does not modify the number of hops remaining in the other dimensions to reach the packet destination.

This kind of topologies is known as k–ary n–cubes. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_d–ary n_d–cubes[1], where k_d is the number of nodes in each of the n_d dimensions in a direct network. The total number of processing nodes in the system is given by $N = {k_d}^{n_d}$. In these topologies, nodes are labeled by an identifier with as many components as topology dimensions $\{p_{n_d-1}, \ldots, p_0\}$, and the component associated to each dimension ranges from 0 to k_d-1 (i.e., nodes are numbered from $\langle 0, 0, \ldots, 0\rangle$ to $\langle k_d-1, k_d-1, \ldots, k_d-1\rangle$). The identifiers of neighbor nodes in a given dimension only differ in the component corresponding to that dimension, while the remaining components have the same value. For instance, two nodes p y p' are neighbors in the x dimension, if and only if $p_x = p'_x \pm 1$ and $p_i = p'_i$ for the rest of dimensions.

The most commonly–used direct topologies are mesh, torus and hypercube. In a mesh topology, all the nodes of a dimension compose a linear array. In torus, all the nodes of each dimension form a ring. The hypercube is a particular case of a mesh where there are only two nodes in each dimension ($k_d = 2$), which forces the number of dimensions (n_d) to be large enough to interconnect all the nodes of the system (i.e., $n_d = log_2 N$). Direct topologies have been used in several of the most powerful supercomputers, being the 3D-torus the most widely used one. For instance, the number two (Titan) and number three (Sequoia) of the November 2015 Top500 supercomputer list [6] use a torus.

For a given number of processing nodes N, direct topologies provide a richer connectivity as the number of dimensions increases. As the hypercube has the largest possible number of dimensions, it provides a better connectivity than meshes and tori, but at a higher cost, since it uses more links and it has a higher router degree (number of ports of the router).

Latency is related to the average distance, measured as the number of links that packets have to cross to reach their destination. Related to this, the diameter of a topology measures the maximum distance between two nodes of the topology using the shortest possible path between them. For a constant number of processing nodes ($N = k_d^{n_d}$), diameter is increased as the number of dimensions (n_d) is decreased and the number of nodes per dimension (k_d) is increased. The increase of the distance traversed by packets also increases the probability of contention with other packets that are also crossing the network, which reduces network throughput. From this point of view, it may seem interesting to maximize the number of dimensions of direct topologies for a given number N of processing nodes. Nevertheless, other issues have to be

[1] The subscript "$_d$" stands for direct.

considered. Direct topologies up to three dimensions can be implemented by using relatively short links in the 3-D physical space and with an acceptable wiring complexity, regardless of the size of the system. However, implementing a topology with more than three dimensions implies increasing the length of the links, since we have to map all the dimensions to the 3-D physical space. In addition, they also require using routers with a higher number of ports (two bidirectional ports per each dimension are required, one per each direction in that dimension). So, direct topologies have some limitations for the implementation of large-scale machines, since they would require using a large number of nodes per dimension (k_d), which increases the diameter of the network and also the probability of contention which increases message latency of communications and reduces network throughput.

The alternative is to use indirect topologies. In these topologies, processing nodes do not have routing capabilities, since the router is separated from the processing node[2]. Routers become independent devices, known as switches. Processing nodes are connected to a switch of the network. Moreover, opposite to direct topologies, all the switches have not necessarily an associated processing node. In fact, most of the switches are usually connected to other switches but they are not connected to processing nodes.

The most common indirect networks are multistage interconnection networks (MINs). In MINs, switches are organized as a set of stages. Processing nodes are connected to the first stage, and the other stages are connected among themselves using a certain connection pattern that provides full–connectivity among all processing nodes. Two different types of MINs can be defined. Unidirectional MINs (UMINs) use unidirectional switches and links, so the network is traversed by packets only in one direction. In this case, processing nodes are also attached to the last stage and a unique path between each source–destination pair is provided (using the minimum number of stages). Bidirectional MINs (BMINs) use bidirectional links and switches. So, in order to travel from a source to a destination processing node, packets travel upwards the network and then downwards. BMINs provide several paths between each source–destination pair.

In MINs, connection patterns between stages are based on permutations of the identifiers of the processing nodes. Depending on the connection pattern used among adjacent stages, several MINs have been proposed. The most widely–used MIN in commercial systems is the fat–tree topology [25], which is a BMIN. However, there are other more recent systems, like the one proposed in [8], focusing on fault tolerance. Most of the high–performance interconnect vendors as Mellanox (manufacturer of the Infiniband technology) [3], Myricom (manufacturer of Myrinet) [4] or Quadrics (manufacturer of QsNet) [5] recommend to use a fat–tree and provide specific switches for building this topology. Moreover, it has been used in some of the most powerful supercomputers. For instance, the Tianhe 2 and the Tianhe 1A supercomputers, the

[2] Direct topologies with independent routers are also possible.

Table 1 Parameters of the different analyzed topologies.

Topology	Diameter	Bisection bandwidth
Mesh	$n_d(k_d - 1)$	$2N/k_d$
Torus	$n_d(k_d/2)$	$4N/k_d$
Fat–tree	$2n_i$	N

number one and twenty six respectively in the November 2015 Top500 list [6] use this topology.

The k–ary n–tree is the most widely–used implementation of the fat–tree topology. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_i–ary n_i–tree[3]. k_i is the number of links that connect a switch to the next or the previous stage, and n_i is the number of stages of the indirect network. So, each switch has $2k_i$ bidirectional ports in fat–trees (i.e. using switches with $d \times d$ ports, k_i is $d/2$). A fat–tree requires at least $log_{k_i}N$ stages to interconnect N processing nodes. Each stage has N/k_i switches, with a total of $(N/k_i)log_{k_i}N$ switches and $N = {k_i}^{n_i}$ processing nodes.

In fat–trees, the diameter of the network depends only on the number of stages, and it is given by $2n_i$, that is $2log_{k_i}N$, as in the worst case a packet must traverse upwards all stages and all stages downwards. Notice that, for a UMIN, the distance traversed by packets is always n_i, regardless of the source and destination processing nodes. In summary, for a fixed number $N = {k_i}^{n_i}$ of processing nodes of a MIN, when the number of stages n_i is increased, k_i is decreased but the distance that packets have to traverse to reach the destination is increased. Also it should be taken into account that, by using high-degree switches fewer switches will be required, but each of them will be more complex. On the contrary, if we use low–degree switches, we will require more switches, but much simpler.

Table 1 shows the diameter and bisection bandwidth for both types of topologies. As it can be seen, the bisection bandwidth is larger for indirect topologies –it depends only on N, without being divided– and the diameter of indirect topologies depends only on the number of stages while in direct topologies it depends on the product of n_d and k_d, which will lead in practice to larger diameters. In direct topologies, for a low number of network dimensions (remember that it is limited by the 3D physical space), the number of nodes per dimension will be high, negatively impacting diameter and bisection bandwidth. Indirect networks provide better scalability than direct networks, because they provide smaller diameters and shorter latencies for large network sizes. Nevertheless, they have a higher cost, because they require a high amount of switches and links, and their physical implementation is costly since the complexity of network wiring grows with its size, unlike direct topologies.

[3] The subscript "$_i$" stands for indirect.

3 The KNS Family of Hybrid Topologies

This paper proposes a new family of topologies for interconnection networks that allows to efficiently connect an extremely high number of processing nodes, given the huge size of current and near future supercomputers [6]. We propose an hybrid topology based on combining an n–dimensional direct network with small indirect topologies. The aim is to combine the advantages of direct and indirect topologies to obtain a family of topologies that is able to connect a high number of processing nodes, providing low latency, high throughput and a high level of fault–tolerance at a lower hardware cost than indirect networks. In the proposed topology, the nodes of each dimension are connected through an indirect topology that allows to have a large number of nodes per dimension without negatively affecting performance.

The new family of topologies will use two different kind of switches in the network (although this is not entirely true, as we will see later). First, low–degree switches are used to connect processing nodes to each dimension and move packets across dimensions. We will refer to these switches as *routers*. They have, at least, one processing node attached to them and as many ports as dimensions. Nowadays, it is common that processing nodes are composed of a large number of cores, and the core count per processing node trend is to increase even further. So, these processing nodes need network interfaces with high bandwidth to avoid bottlenecks in the end processing node connection to the network. In fact, there are already some commercial solutions which use network interfaces cards with dual ports [1]. Considering the core count increase trend it is expected that the bandwidth requirements of end processing nodes will be even higher. Another key point of these network interfaces with several ports is that they provide fault-tolerance. Using one port network interfaces causes having a single point of failure that will disconnect a high number of cores. Both, bandwidth and fault tolerance requirements will provoke increasing the number of ports in the near-future network interfaces.

In this paper we take advantage of these network interface cards with several ports to implement the routers used in the new family of topologies.

Additionally to these routers, the proposed topology also uses other *switches* to implement the indirect networks that interconnect the routers of each dimension.

3.1 Description of the family

The newly proposed topology family arranges processing nodes and their associated routers (node) in n dimensions like a direct topology. But, contrary to mesh and torus topologies, routers of a given dimension are not only connected with their adjacent routers in that dimension. Instead, all the routers of a given dimension are connected by means of a small indirect network. This indirect network could be even a single switch, considering the number of ports of current commercial high-radix switches. However, if the number of routers

per dimension exceeds the number of ports of the available switches, the indirect network will be arranged as a small MIN. We will refer to this MIN as the *indirect subnet*. This will provide a low latency communication among routers in the same dimension with a small hardware extra cost compared to direct topologies. In this way, the number of routers per dimension stops being a bottleneck from the point of view of the performance, as the time to communicate two routers of the same dimension is constant or grows logarithmically with the number of routers per dimension. Notice also that each router only requires a bidirectional link per dimension to connect to the switch of each dimension. On the contrary, a mesh or torus require two bidirectional links per dimension, one per neighbor node.

The proposed family of topologies is defined by three parameters. Two of them are inherited from direct networks: the number of dimensions n_h[4] and the number of routers per dimension k_h. The number of processing nodes it can interconnect is given by $N = k_h^{n_h}$. In addition to these two parameters, there is an additional parameter, the number of stages of the indirect subnet, referred to as s_h. This number depends on the number of routers per dimension k_h and on the number of ports of the switches used to implement the indirect subnet (which will be referred to as d_h). The ratio between k_h and d_h defines the way to interconnect the k_h routers of each dimension. If $k_h \leq d_h$, a simple switch would be able to interconnect all the routers of the dimension, and s_h will be equal to 1. On the contrary, if $k_h > d_h$, a MIN would be required to interconnect the routers of each dimension, and the number of required stages will be given by $log_{\frac{d_h}{2}} k_h$ (remember that in an indirect network built with d_h-port switches, the network radix is equal to $\frac{d_h}{2}$). We will refer to the new family of topologies proposed in this paper as k_h–ary n_h–direct s_h–indirect (KNS), where k_h is the number of routers per dimension, n_h is the number of dimensions and s_h is the number of stages of the indirect subnets.

In this paper, we have considered two different MINs to connect the routers of a given dimension. The first one is a BMIN, the fat–tree. The second one is RUFT [19], a UMIN derived from a fat–tree using a load–balanced deterministic routing algorithm [18], which requires less hardware resources.

Figure 1 shows an example of the new topology with 2 dimensions ($n_h = 2$), and 4 routers per dimension ($k_h = 4$), with a total number of 16 routers. In this case, the routers of the same dimension are connected by a single 4-port crossbar. However, for a higher number of routers per dimension, say $k_h = 8$ and the same switch size, a MIN should be used. In the case of using fat–tree subnets, for interconnecting the 8 routers of each dimension, it will require 3 stages and 12 bidirectional 4-port switches, implementing in this way a 8–ary 2–direct 3–indirect. In the case of using RUFT as indirect subnets, it is also a 8–ary 2–direct 3–indirect but built with 4-port unidirectional switches. We will analyze in more depth the issues of using different indirect subnets in Section 5.

[4] The subscript "h" stands for hybrid.

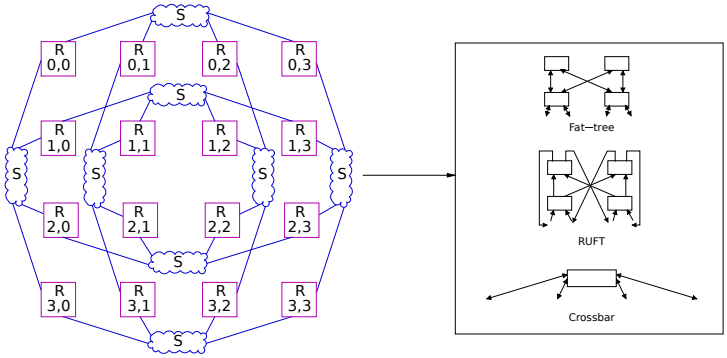

Fig. 1 An example of the KNS topology with $n_h = 2$, $k_h = 4$ and $d_h = 4$.

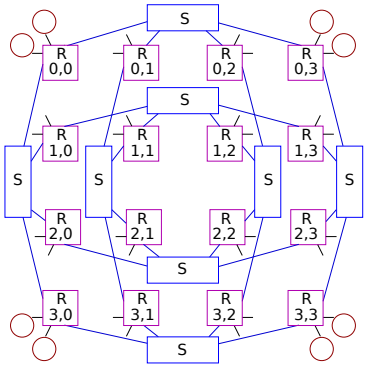

Fig. 2 An example of the KNS topology with $n_h = 2$, $k_h = 4$, $s_h = 1$ and $p_h = 2$.

As it can be seen in Figure 2, the routers are connected to all dimensions through a different link (one per dimension). Notice that it is also possible to attach several processing nodes to the same router. Having more than one processing node attached to each router is known as concentration in the literature. In fact, the topology shown in the figure already has two processing nodes attached to each router. Only the corner processing nodes are shown in the figure for the sake of clarity. This introduces a new parameter on the topology family, p_h, the number of processing elements that are connected to each router. In this case, the number of processing nodes is given by $N = p_h k_h^{n_h}$.

In Figure 1, two different components with switching capabilities can be distinguished. The switches[5] (in blue and labelled with a "S" in Figure 1) are only connected to other switching components, whereas the routers (in pink and labelled with an "R" in Figure 1) are used to connect the processing nodes to the network through several dimensions. These routers are connected

[5] It may be a single switch or a set of switches forming a MIN.

Table 2 Parameters of the different analyzed topologies.

Topology	Parameters	
Mesh or Torus	n_d	# dimensions
	k_d	# of nodes per dimension
Fat–tree	n_i	# of stages
	k_i	switch arity
k_h–ary n_h–direct s_h–indirect (KNS)	n_h	# of dimensions
	k_h	# of routers per dimension
	s_h	# of stages of the indirect subnet
	d_h	switch degree
	p_h	# processing nodes per router

on one side with processing nodes and on the other side with switches, and their degree is $n_h + p_h$, that is, the sum of the number of dimensions and the number of concentrated processing nodes. As it can be seen in Figure 1, if a crossbar is used as indirect subnet, switches allow packets to change to any position in a given dimension by traversing only two links, whereas routers allow to travel between dimensions. Thus as, at most, only two hops are performed per dimension, the diameter of the new topology is $2n_h$, which is a very low value. If a MIN is used instead of a crossbar as indirect subnet, the diameter of the network will be the diameter of the used MIN (for a fat–tree, i.e. a BMIN, 2× the number of stages, which is expected to be a small number due to using high-radix switches) multiplied by n_h, that is $2s_h * n_h$.

Table 2 summarizes the parameters that define the hybrid family of topologies. Notice that only one of d_h or s_h parameter is necessary, as the another one can be derived from the other parameters. In addition, the parameters that define traditional direct and indirect topologies are also shown.

The proposed hybrid topologies can take advantage of the high number of ports available in current switches. For example, edge switches of up to 36 ports are commercially available [2], while chassis switches offer up to 648 ports [2]. With such switches, in most cases, a small MIN with only 2 or 3 stages or even a single switch will be enough to connect the routers in each dimension. As the MIN is very small, it will introduce low latency and it can be easily implemented with low wiring complexity.

On the other hand, as already stated, this topology can take advantage of new network interfaces (like the new commercial HCA cards [1]), that have several ports to connect the processing nodes to the network. By using these network interfaces, the proposed topologies can be implemented by integrating the router into the processing node as part of the network interface. These network interfaces will have switching capabilities, so processing nodes equipped with these new network interfaces will be able, apart from injecting messages in the network, to route packets that are not destined to them to other processing nodes without ejecting messages from the network.

The resulting topologies could seem similar to BCube [20] or Hypercrossbar network [10], but the great difference between our proposal and BCube is that the processing nodes do not eject messages from the network; the new topology

family forwards the messages in the network interface which highly reduces latency. However, these cases could be considered as particular configurations of the KNS family of topologies.

The topologies proposed in this paper have several main advantages. First, they allow to highly reduce the diameter compared to direct topologies. This will lead to network performance improvements, decreasing latency and increasing network throughput. Additionally, the number of required switches and links is reduced compared to an indirect topology that connects the same number of processing nodes, as will be shown in section 5. Therefore, it is expected that the proposed family of topologies reduces the cost of the network. Finally, it also provides a good fault–tolerance level (see Section 5).

4 Routing Algorithms for the KNS Family of Topologies

In this section, we describe the routing algorithms proposed for the new family of topologies. We will first describe the ones proposed for k_h–ary n_h–direct 1–indirect topologies (i.e., a crossbar is used as indirect subnet). Then, we describe the ones proposed for the general case, that is, for KNS topologies, using a fat–tree or a RUFT as indirect subnets.

First, we explain how routers and switches are labeled in the KNS topology. Each router is labeled as in meshes and tori, with a set of components or coordinates (as many as network dimensions) $\langle r_{n_h-1}, r_{n_h-2}, \ldots, r_1, r_0 \rangle$. Each coordinate represents the position of each router in each of the dimensions. On the other hand, the switches are labeled by a 2-tuple *[d,p]*, where d is the dimension the switch is located at, and p is the position of that switch in that dimension. Notice that routers do not belong to any dimension, since they are connected to all of them, and packets change dimensions through them. On the contrary, switches do not allow changing the dimension packets are traversing, they just allow packets to move through that dimension.

4.1 Routing in k_h–ary n_h–direct 1–indirects

Although both deterministic and adaptive routing algorithms could be used, taking into account that adaptive routing may introduce out-of-order delivery of packets and that leads to a more complex implementation, in this paper, we will focus only on deterministic routing.

The deterministic routing algorithm for k_h–ary n_h–direct 1–indirect topologies, which will be referred to as Hybrid–DOR, is a variation of the dimension ordered deterministic routing algorithm (DOR) for meshes, adapted to the k_h–ary n_h–direct 1–indirect topology. In DOR, packets are routed through the different dimensions following an established order until the destination processing node is reached. At each dimension of the mesh, packets traverse several routers until the movement in that dimension is exhausted. On the other hand, as each mesh router has two links per dimension, packets must

be forwarded in each dimension through the direction that guarantees the minimal path.

In Hybrid–DOR, network dimensions are also crossed in an established order to guarantee deadlock freedom, as in DOR. However, there is a unique link per dimension that connects the current router to a switch that allows directly reaching any of the processing nodes in that dimension. So, packets do not perform several hops at each dimension. Instead, in Hybrid–DOR, routers directly forward packets through the unique link of the dimension they have to traverse, and this link is connected to the corresponding crossbar that moves the packet to the destination component in that dimension. Notice that, contrary to meshes and tori, in k_h–ary n_h–direct 1–indirect topologies, it is not required to choose the direction at each dimension, as there is only one link per dimension. The routing in the switches is very straightforward since, they just must forward packets through the link indicated by the destination component in the current dimension, requiring just one hop to reach next router.

Next, we show the Hybrid–DOR pseudo–code for the routers and the crossbars of the network. The number of dimensions of the topology is n_h and the destination and current router coordinates are given by $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, and $\langle r_{n_h-1}, \ldots, r_{d+1}, r_d, r_{d-1}, \ldots, r_1, r_0 \rangle$, respectively. In the case of crossbars, the current switch is given by *[d,p]* (the p^{th} switch of the d dimension). The chosen link to send the packet is returned in $link$.

Routers:

$i = 0;$
$Done = False$
while $(i < n_h) \wedge (!Done)$ **do**
  if $x_i != r_i$ **then**
    $Done = True$
    $link = i$
  end if
  $i = i + 1$
end while

Crossbars:

$link = x_d$

As can be seen, routers select the next dimension to forward the packet, which it is also the link of the router to be used, since there is just one link per dimension, and crossbars merely select the link given by the destination coordinate of the corresponding dimension to reach the destination component in that dimension.

4.2 Routing in k_h–ary n_h–direct s_h–indirects

In k_h–ary n_h–direct s_h–indirect topologies, the crossbars are replaced by small MINs. As stated above, the MINs considered in this paper are fat–trees or RUFTs. In these topologies, all the switches of a given fat–tree or RUFT are always in the same dimension and in the same position relative to the routers. In order to identify the switches inside a given fat–tree or RUFT, we extend the classical switch coordinates from MINs by including the coordinates of

the MIN in the direct topology. In this way, the switch coordinates in k_h–ary n_h–direct s_h–indirect topologies will be given by a 4-tuple *[d,p,e,o]*, where d is the dimension the MIN belongs to, p is the position of the MIN in that dimension, e is the stage of the switch inside the MIN, and o is the order of that switch in that stage. Remember that d and p are the coordinates of the equivalent crossbar in k_h–ary n_h–direct 1–indirect topologies.

Since the routers are the same regardless of the indirect topology used, its routing algorithm is the same as the one shown for k_h–ary n_h–direct 1–indirect topologies. However, switch routing algorithm depends on the particular indirect network used.

First, we focus on the k_h–ary n_h–direct s_h–indirect topology that uses fat–trees in the indirect subnets. Despite the fact that a fat–tree has several paths for each source–destination pair (i.e., it allows adaptive routing), we propose to use the deterministic routing algorithm presented in [18] since it is simpler and is able to outperform adaptive routing. We will summarize that routing algorithm here. Routing is composed of two subpaths. First, packets are sent upwards to the common ancestor switch of the source and destination processing nodes and, then, they are sent downwards to its destination. Traffic is balanced by carefully selecting the links to be used according to the destination processing node. In particular, the link to be used in both subpaths is given by the destination coordinate corresponding to the stage where the switch is located at. For instance, if a switch located at stage e_1 routes a packet whose destination (in the fat–tree) is $\langle t_{n_i-1}, \ldots, t_{e_1+1}, t_{e_1}, t_{e_1-1}, \ldots, t_1, t_0 \rangle$, then the packet is sent through the link $k_i + t_{e_1}$ in the upwards phase and through link t_{e_1} in the downwards phase. Remember that k_i is the arity of the switches of the fat–tree topology. Please see [18] for more details.

In the fat–trees subnets of the k_h–ary n_h–direct s_h–indirect topologies, the routing algorithm is the same, but only the part of the destination identifier corresponding to the dimension the fat-tree belongs to (i.e., x_d in our notation) is used. In this way, the packet is delivered to the same router that would be reached through the corresponding crossbar in a k_h–ary n_h–direct 1–indirect topology.

This routing algorithm is shown below. Assume that destination coordinates are $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, the dimension where the fat–tree is located at is d, $GetFTIdentifier$ returns from x_d the fat–tree coordinates to route locally in the fat–tree, $UpwardsPhase$ returns *true* if the packet is in its upwards subpath, or *false* otherwise, and that the stage in the fat-tree of the switch that is routing the packet is given by e:

Switches:
$t = GetFTIdentifier(x_d)$
if $UpwardsPhase()$ **then**
  $link = k_i + t_e$
else
  $link = t_e$
end if

Let us consider the case where RUFT is used in the indirect subnets of the KNS topology. In RUFT, there is a unique path between each source–destination pair and packets have to cross all the stages, reaching the last stage, which is directly connected back to the processing nodes. The link to be used by a packet at a particular switch is given by the destination component corresponding to the stage the switch is located at. Please see [19] for details. In this case, the pseudo–code for the switch routing algorithm is the following:

Switches:

$t = GetFTIdentifier(x_d)$

$link = t_e$

If Hybrid-DOR is used in the routers jointly with the aforementioned algorithms for the switches in the indirect networks, the resulting routing algorithm for the new topology is deterministic and deadlock-free, since dimensions are crossed in order in the direct topology and the routing algorithm used in the indirect networks has not any loop in its channel dependency graph [14].

5 Evaluation

In this section, we evaluate the KNS topology family, comparing it with other topologies such as meshes, tori, fat-trees, and flattened-butterflies [22].

5.1 Network Model

To evaluate the family of topologies proposed above, a detailed event–driven simulator has been implemented. The simulator models several topologies, including the new family of topologies presented in this paper, the KNS. This simulator uses virtual cut–through switching. Each switch has a full crossbar with queues of two packets both at their input and output ports. Credits are used to implement the flow control mechanism. We assumed that it takes 20 clock cycles to apply the routing algorithm; switch and link bandwidth has been assumed to be one flit per clock cycle; and fly time has been assumed to be 8 clock cycles. These values were used to model Myrinet networks in [16]. In addition, for the new topology using RUFT as indirect subnet, the fly time of the long links that connects the output of the unidirectional MIN to the direct routers is assumed to be 8 clock cycles per stage, in order to take into account that these links are longer.

We have performed the evaluation by using several synthetic traffic patterns: uniform, hot–spot, tornado, and complement. In the uniform traffic pattern, message destination is randomly chosen among all destinations. In the hot–spot traffic pattern, a percentage of traffic is sent to a small subset of the processing nodes (5% of nodes in this case) and the rest of the traffic is uniformly distributed. In complement, the destination processing node is obtained by complementing all the component bits of the source processing node. Therefore, in this traffic pattern, the destination processing node of all

packets generated at a given source processing node is always the same. In tornado [28], the destination is chosen in such a way that each packet travels $n(\frac{k}{2}-1)$ hops. Regarding packet size, the results shown in this paper have been obtained for 256–flit packets. However, simulations with other packet sizes, such as 16, 128, and 512 flits has been performed, and the results are consistent with the ones shown here. For each simulation run, the full range of injected traffic (from low load to saturation) has been tested.

5.2 Performance Results

In this section, we compare the KNS using different indirect subnets (crossbar, fat-tree, and RUFT) against other well–known and frequently–used topologies such as tori, meshes, and fat–trees. Moreover, we also compare our proposal against the flattened–butterfly (FB) topology because it is becoming a popular topology in recent research papers (see Section 6 for further details). The FB topology is a variation of the butterfly topology obtained from using high-radix switches, that results in a direct topology. This topology can be seen as a generalized hypercube with concentration, as all the switches in the same dimension are directly connected, that is, there is a link from each switches to the others of the same dimension. Several FB configurations have been tested and compared to our proposal.

We have evaluated a wide range of network sizes, from 64 to 64K processing nodes. Larger topologies have not been simulated due to simulator memory constraints. For direct topologies, we have tested different values of the number of dimensions and number of nodes per dimension. In particular, we have evaluated networks of 2 dimensions, with 4, 8, 16, 32, 64, and 256 nodes per dimension; three dimensions, with 4, 8, and 16 nodes per dimension; four dimensions, with 4, 8, and 16 nodes per dimension; six dimensions, with 4 processing nodes per dimension; and eight dimensions, with 4 nodes per dimension. If not stated the contrary, only one processing node is attached to each router (i.e. without concentration). If several ones are attached, the x–p suffix is used, x being the number of processing nodes attached to each router. These networks are compared with fat–trees and FBs with the same number of processing nodes. Notice that, in some cases, several configurations are possible. For the sake of clarity, only a subset of the most representative simulations is shown.

Uniform traffic pattern: Figure 3.(a) shows results for 2-D small networks (256 processing nodes) with uniform traffic. As it can be seen, the mesh is the network that achieves the lowest throughput, followed by torus and the FBs configurations. In this latter case, we have selected a 4–ary 3–cube and a 2–ary 7–cube FB with concentration in order to compare our proposal against a topology with a hardware of similar complexity, as we will show later (in Section 5.3). The next topologies that achieve a better performance are the two fat–tree configurations. However, the best absolute throughput is achieved by the family of topologies proposed in this paper. In particular, the different

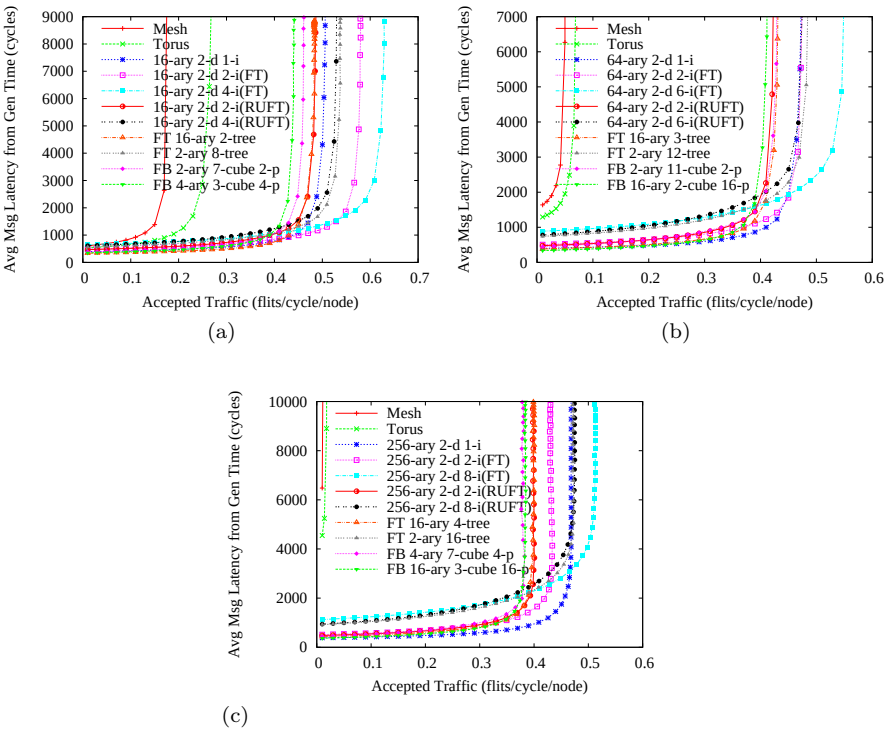

Fig. 3 Average packet latency from generation vs. accepted traffic for uniform traffic and 2 dimensions for direct topologies. (a) 256 processing nodes. (b) 4K processing nodes. (c) 64K processing nodes.

tested configurations ordered from lower to higher throughput are the topology that uses a RUFT with two stages as indirect subnet (16–ary 2–d 2–i (RUFT)), the ones that uses crossbar (16–ary 2–d 1–i), the one that uses RUFT with 4 stages (16–ary 2–d 4–i(RUFT)), the one that uses a FT with two stages (16–ary 2–d 2–i (FT)) and the one that uses a FT with four stages (16–ary 2–d 4–i (FT)). In particular, the best configuration of the new topology family obtains 3 times more throughput than the worst network (mesh), more than twice versus torus, more than 20% versus FT and about 40% improvement versus FB.

Figures 3.(b) and 3.(c) show how throughput is decreased in all the topologies as we increase the number of processing nodes in the network, keeping constant the number of dimensions in the direct topologies, and therefore increasing the number of routers (processing nodes) per dimension. In the case of mesh and torus topologies, throughput strongly decreases, as the average distance between two nodes is markedly higher than in the other topologies. Regarding the KNS topologies, all the tested configurations outperform both the FT and FB configurations analyzed. The best configurations are again

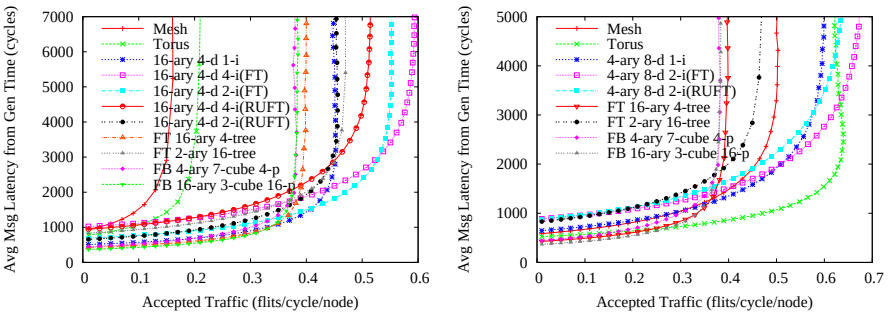

Fig. 4 Average packet latency from generation vs. accepted traffic for uniform traffic with 64K processing nodes and different number of dimensions: (a) 4D and (b) 8D.

the ones that use the tallest FT as indirect subnet. Notice that the different topologies have a different hardware cost, which is evaluated in following sections.

In Figures 3.(a), 3.(b), and 3.(c) we can also see the impact of using more stages in the MINs of KNS topologies. For the same number of routers per dimension, if we decrease the number of stages, the arity of the switches is increased, and a lower latency should be obtained. The plots show that, the higher the number of stages, the higher the base latency (in more detail the zero-load latency) as more switches have to be crossed by packets. Surprisingly, networks with more stages also achieve more throughput. This effect is explained by the reduction of the head-of-line (HoL) blocking effect. For a given number of routers per dimensions, a taller FT uses smaller switches (i.e. with lower number of ports). As a consequence, each switch port is potentially demanded by a lower number of input ports and, hence, the effect of HoL blocking is reduced. From another point of view, with fewer stages, each indirect topology has less switches to serve the same number of routers. Thus, each switch has to deal with more traffic, leading to more HoL blocking effect and, hence, less throughput.

Let us analyze the base latency. In a k_h–ary n_h–direct 1–indirect, base latency does not depend on the number of routers per dimension. However, in KNS that uses MINs, the base latency increases with the number of processing nodes because the number of stages in the indirect subnets also grows in order to connect a larger number of routers. This effect is more prominent in RUFT, due to the fact that packets traverse always all stages since it is a UMIN topology. In the case of torus and mesh, base latency strongly depends on the number of nodes per dimension, as average distance between nodes is increased.

Figure 4 analyzes the impact of the number of dimensions in the different topologies. We analyze a network with 64K processing nodes implemented with a different number of dimensions. We can distinguish three different behav-

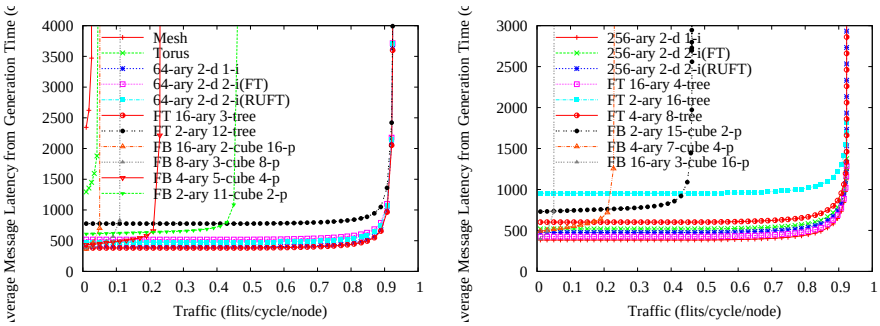

Fig. 5 Average packet latency from generation vs. accepted traffic for complement traffic and (a) 4K processing nodes and (b) 64K processing nodes.

iors. First, Mesh and torus topologies have a similar behavior. The higher the number of dimensions, the fewer the number of nodes per dimension, and the higher the achieved throughput. Also, base latency decreases with the number of dimensions, because the average distance is reduced. However, the behavior of k_h–ary n_h–direct 1–indirect is different. Throughput also increases with the number of dimensions because the size of switches of the indirect network (a crossbar in this case) is reduced, and, hence, the pernicious effect of HoL blocking is reduced. However, base latency does not improve with the number of network dimensions. This is due to the fact that the number of hops that packets must perform also grows with the number of dimensions. Concerning the k_h–ary n_h–direct s_h–indirect, they have a similar behavior to the previous one, but with a difference. The base latency, in this case, slightly decreases when the number of dimensions increases. Although network diameter increases with the number of dimensions, as started above, as there are fewer routers per dimension, indirect subnets have fewer stages, and, thus, packets have less stages to cross. Finally, the configurations of the FB shown (the ones with a hardware cost similar to the one of our proposal, see Section 5.3) and FT obtain an intermediate throughput value. Anyway, we would like to remark that the new family of topologies always obtains the best throughput regardless of the number of dimensions.

Complement traffic pattern: Figure 5 shows the obtained results for this traffic pattern for different networks (4K and 64K processing nodes). We can distinguish two different behaviors among the analyzed topologies. In torus, mesh, and FB topologies, the network is rapidly saturated. In the rest of topologies (all the KNS topologies, and fat–trees), the network is able to cope with all the injected traffic. The reason is that, for this traffic pattern, as an optimal load-balanced routing algorithm [18] is used in FT and RUFT, the network resources are not shared among source–destination pairs. The same happens in the indirect subnets of the proposed family of topologies as they use the same routing scheme, and, in addition, links connecting

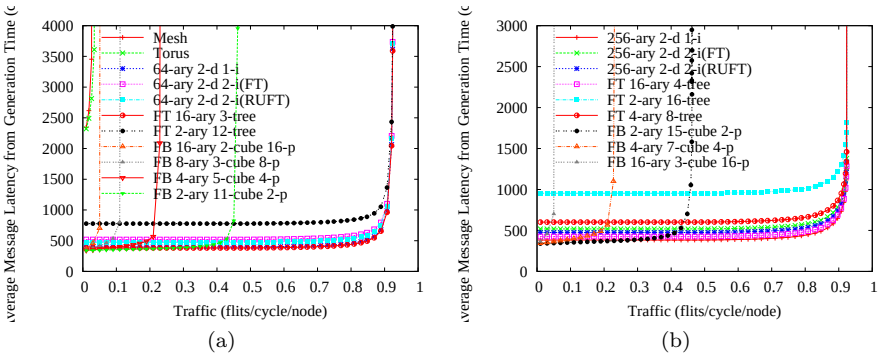

Fig. 6 Average packet latency from generation vs. accepted traffic for tornado traffic and (a) 4K and (b) 64K processing nodes.

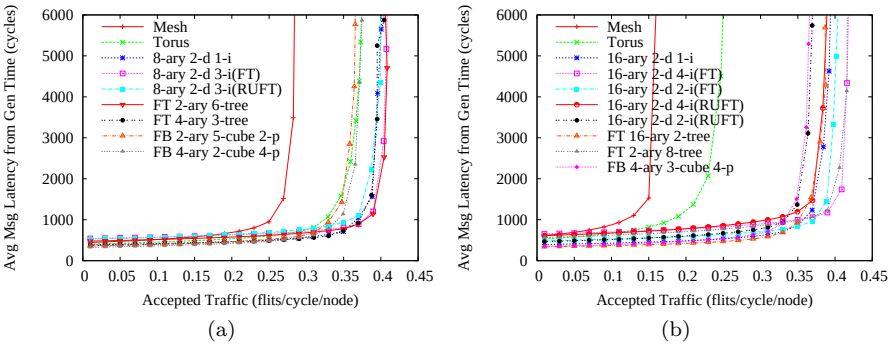

Fig. 7 Average packet latency from generation vs. accepted traffic for Hot–Spot traffic at 5% in 2 dimensions. (a) 64 processing nodes. (c) 256 processing nodes.

routers and switches only forward packets between a source and a destination, since each router only has one processing node attached to it and a source processing node only generates traffic to a given destination. Thus, the path used by packets from a given source-destination pair is not shared with any other packets destined to another processing node. So, hybrid topologies and fat–trees are clearly the winners for this traffic pattern, and direct topologies are not a good option.

Tornado traffic pattern: For this traffic pattern, shown in Figure 6, we obtain similar results than in complement traffic pattern. The KNS topologies and fat–trees are able to cope with all the injected traffic. Again, this very good behavior is due to the use of a load-balanced routing algorithm and the higher effective bisection bandwidth of these topologies.

Hot–spot traffic pattern: As expected, the concentration of packets sent to a few processing nodes makes the network saturate at a lower throughput

Table 3 Analytical comparison of the Mesh, Torus, Fat–Tree, Flattened–Butterfly and the KNS topologies. KNS topologies refers to the arity of indirect switches.

	Mesh	Torus	Fat–tree	Flattened–Butterfly
Switches	N	N	Nn_i/k_i	$k_f^{n_f}$
links	$(k_d-1)k_d^{n_d-1}n_d+N$	$k_d^{n_d}n_d+N$	$N(n_i-1)+N$	$k_f^{n_f}(k_f-1)n_f/2+N$

	KNS
Switches	$N/p_h(s_h n_h/k_i)$
Routers	N/p_h
links	$k_h^{n_h}n_h+(s_h-1)Nn_h/p_h+N$

than in other traffic patterns. As it can be seen in Figure 7, the worst topology is again the mesh; the torus saturates also slightly after it. The remaining topologies obtain a similar throughput, being the best one the hybrid topology that uses crossbars as subnets.

To summarize, the new family of topologies is able to obtain, in all analyzed traffic patterns and network configurations, equal or better performance results than the direct and indirect topologies evaluated for different traffic patterns.

However, each network topology has different complexity. In the next section, we estimate the complexity and cost of each network. Then, we will perform a comparison of topologies from a cost-performance point of view.

5.3 Cost–performance analysis

This section estimates and compares the hardware cost of each considered topology connecting the cost also with the perfomance of each of them. First, we will analyze, for each topology, the number of links and switches that it requires. However these numbers of links or switches in isolation is not a good metric, since a network with more complex switches (which are more expensive) has a lower number of links and switches and (incorrectly) seems to be cheaper. Therefore, neither the number of links nor the number of switches by themselves are accurate metrics of the actual cost. This is why we will provide another way to mesure the actual cost.

Table 3 shows how to compute the number of links and switches for each topology (it also shows the number of direct routers for the new family of topologies). For example, in k_h–ary n_h–direct 1–indirect (i.e. it uses crossbar as subnets), if we have N processing nodes and p_h concentrated processing nodes per each direct router, we will need N/p_h or, what is the same, $k_h^{n_h}$ direct routers and one switch (crossbar) for each group of k_h routers with the same dimension component. Then, we will need crossbars with k_h ports (in this case k_h is equal to k_i because a single switch is used in each dimension) to connect the k_h direct routers in each dimension, so we will need $N/p_h(n_h/k_h)$ crossbars.

If we use MINs as subnets (fat–tree or RUFT), we will need k_h/k_i switches per stage to implement the MIN that replaces the crossbar, yielding N/p_h

Table 4 Results for different 2–D topologies with uniform traffic and 64K processing nodes.

Topology	Base Latency	Throughput	Links	Switches	Routers (KNS)
256–ary 2–direct 1–indirect	386	0.47	196,608	512	65,536
256–ary 2–direct 2–indirect(RUFT)	480	0.40	327,680	16,384	65,536
256–ary 2–direct 2–indirect(FT)	516	0.43	327,680	16,384	65,536
256–ary 2–direct 4–indirect(RUFT)	635	0.41	589,824	131,072	65,536
256–ary 2–direct 4–indirect(FT)	727	0.48	589,824	131,072	65,536
256–ary 2–direct 8–indirect(RUFT)	941	0.48	1,114,112	524,288	65,536
256–ary 2–direct 8–indirect(FT)	1,129	0.55	1,114,112	524,288	65,536
FB 16-ary 3-cube 16-p	367	0.39	157,696	4,096	0
FB 4-ary 7-cube 4-p	443	0.38	237,568	16,384	0
FB 2-ary 15-cube 2-p	512	0.41	311,296	32,768	0
FT 16-ary 4-tree	428	0.40	262,114	16,384	0
FT 4-ary 8-tree	596	0.41	524,288	131,072	0
FT 2-ary 16-tree	921	0.47	1,048,576	524,288	0
Torus	4,547	0.02	196,608	65,536	0
Mesh	6,478	0.01	196,096	65,536	0

$(n_h/k_h)(s_h k_h/k_i)$ switches with arity k_i, which can be simplified to N/p_h $(s_h n_h/k_i)$ switches.

Regarding links, we will need one link per dimension in each direct router (i.e. $k_h^{n_h} n_h$ links in total) and one link for each processing node to connect it to its corresponding router (N links in total). Furthermore, if we use MINs as subnets (fat–tree or RUFT), each one will have k_h links between each stage (i.e. $(s_h - 1)k_h$ per MIN), yielding $(s_h - 1)k_h N/p_h(n_h/k_h)$ in total, which can be simplified to $(s_h - 1)N n_h/p_h$ links. Notice that all links are bidirectional except those of the RUFT indirect subnets that use unidirectional links, and the same occurs with the switches. In the case of links, their cost is not halved, since much of the cost comes from the connectors. For this reason, and for easier comparison, we assume the same link cost for unidirectional and bidirectional links. Regarding switches, a switch with p unidirectional ports can be implemented by using a switch with $p/2$ bidirectional ports.

Table 4 shows these metrics for different configurations of 64K–processing node topologies including also the performance results for the uniform traffic pattern. The network is 2D in the case of KNS, mesh, and torus topologies. For the FB topology, we have considered different configurations. As it can be seen, fat-trees and the KNS topologies with more stages are the topologies that achieve the highest raw throughput (256–ary 2–direct 8–indirect – FT and RUFT – and FT 2-ary 16-tree). However, if we also consider the cost, these

Table 5 List price for switches: (a) Edge switches and (b) Chassis switches.

Edge Switches	
Ports	List price ($)
12	5,361
18	9,850
36	12,523

(a)

Chassis Switches	
Ports	List price ($)
108	32,650
216	48,778
324	65,875
648	110,177

(b)

Table 6 List price for links: (a) Copper Links and (b) Fiber Links.

Copper Links	
Length(m)	List price ($)
0.5	84
1	94
2	107
3	123
4	139
5	172

(a)

Fiber Links	
Length(m)	List price ($)
3	551
5	551
10	580
15	611
20	642
30	730
50	896
100	1,347

(b)

topologies are composed of a higher number of links and switches. Specially in the case of the 256–ary 2–direct 8–indirect (FT and RUFT) and the FT 2-ary 16-tree. On the other hand, there are topologies that require a smaller number of switches, but these switches have more ports, so they may be more expensive. This is the case of the 256–ary 2–direct 1–indirect or FB 16-ary 3-cube 16-p.

In order to get an actual cost figure, we have calculated the cost (in $) that some of these configurations would have when implemented with real commercial products. We have used InfiniBand products with FDR technology of Mellanox [2] (February 2015) to calculate the cost.

Tables 5.(a) and 5.(b) show the list price of switches depending on their number of ports. There are two different types of switches: edge and chassis switches. When preparing the budget, if there are no switches with the number of ports required by the configuration, we selected the next one with greater number of ports. For example, we needed to use 256-port switches for 256–ary 2–direct 1–indirect, so we selected 324-port switches.

Tables 6.(a) and 6.(b) show the list price of links depending on their length. As copper links are limited to 5 meters, if a longer link is required, fiber links must be used. We assumed an average length between cabinets (global links) of 10 meters, and 2 meters for connections in the same cabinet (local links).

The number of global and local links depends on the topology. For 256–ary 2–direct 1–indirect and 256–ary 2–direct 4–indirect with FT as subnets, the processing nodes of the same first dimension can be placed in the same cabinet

Table 7 Cost for Network Interface Cards.

NIC	List price ($)
Connect IB PCIe 3.0 16x Single Port	1314
Connect IB PCIe 3.0 16x Dual Port	2378

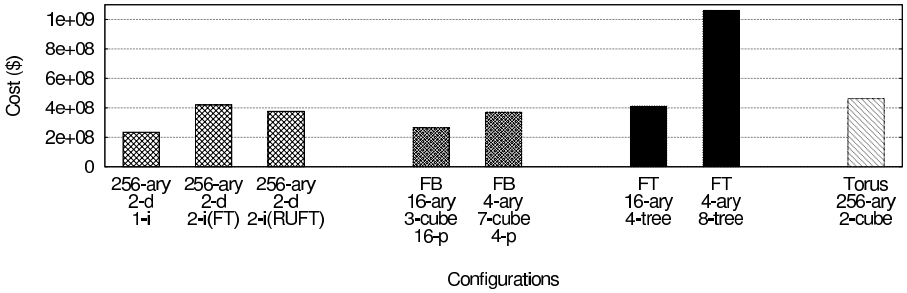

Fig. 8 Total cost of different topology configurations with 64K processing nodes.

or in two cabinets (one beside the other). The links of this dimension are local links, and the links of the second dimension are global links. Regarding the links interconnecting the stages of the MIN (a fat–tree in this case), we assume that they are local, since each subnet fits in a cabinet. In flattened–butterfly configurations, for example FB 16-ary 3-cube 16-p, we use the same approach. The links of the first dimension are local because the processing nodes of the same first dimension are placed in one or two cabinets and the links of the remaining dimensions are global. In fat–trees, the links which connect the processing nodes with the first stage are local, and the remaining links are global. In torus topology, the configuration is very similar to k_h–ary n_h–direct 1–indirect (local links for the first dimension and global links for the second dimension). However, in this case, a cabinet or a group of cabinets that contain processing nodes of the same first dimension, are connected to the neighboring cabinet or group. So, they will be very close. For this reason, in this case we have used shorter global links of 5 meters.

As previously stated, multiport NICs can be used to implement the KNS topologies. There are currently commercially available dual-port NICs that can may used to implement KNS with two dimensions. With these NICs, neither direct routers nor links between processing nodes and direct routers are longer required. The prices of dual–port and single–ports NICs are shown in Table 7.

Notice that all prices shown in Tables 5, 6 and 7 are for individual products. If purchased them massively we will surely enjoy a great discount in all cases.

Using these data, we calculated the cost of some selected configurations, which are shown in Figure 8. The configurations shown are the cheapest ones that their perfomance is not very far to the configuration that obtains the highest throughput.

Table 8 Cost–performance analysis for different topology configurations with 64K processing nodes. Throughput is measured in flits/cycle/node. Throughput/cost is measured in flits/cycle/node/$.

Topology	256-ary 2-direct 1-indirect	256-ary 2-direct 2-indirect (FT)	256-ary 2-direct 2-indirect (RUFT)	FB 16-ary 3-cube 16-p	FB 4-ary 7-cube 4-p	FT 16-ary 4-tree	FT 4-ary 8-tree	Torus 256-ary 3-cube
Throughput	0.47	0.43	0.40	0.39	0.38	0.40	0.41	0.02
Total Cost ($)	235 M	420 M	376 M	266 M	371 M	412 M	1,062 M	463 M
Throughput /Cost	2.00 x10^{-9}	1.02 x10^{-9}	1.06 x10^{-9}	1.47 x10^{-9}	1.02 x10^{-9}	0.97 x10^{-9}	0.39 x10^{-9}	0.04 x10^{-9}

As can be seen in Figure 8, the 256–ary 2–direct 1–indirect configuration obtains the lowest absolute cost. The fat–tree with 8 stages (FT 4-ary 8-tree)has a very high cost, despite having very good performance. In the case of torus, its cost is not very high, but it has a low performance. Flattened–butterfly has a competitive cost, but it is not lower than the 256–ary 2–direct 1–indirect and it does not reach a better perfomance.

To allow a better comparison of both cost and perfomance, Table 8 shows the ratio between cost and perfomance. If we use KNS topologies, configurations with more stages have better throughput but also a higher cost and more latency. The same applies to fat-trees. The k_h–ary n_h–direct 1–indirect combines a good perfomance with a low cost. Although the torus configuration is not very expensive, it has a very poor perfomance. Flattened–butterflies are not very expensive and obtain good perfomance. However, the k_h–ary n_h–direct 1–indirect configuration obtains the best absolute results in terms of perfomance-cost ratio. As it can be seen, the worst ratio is provided by the torus, followed by a configuration of the fat–tree (FT 4-ary 8-tree).

5.4 Fault–tolerance

The proposed topologies provide a lot of alternative paths for each source–destination pair, which is very important to tolerate faults. In this section, we will briefly analyze the fault-tolerance properties of the new topology.

In meshes, the worst case arises when a link connected to a corner node fails. As each corner node has a number of links equal to the number of network dimensions, the maximum number of faults that keeps the network connected is equal to the number of dimensions minus 1 ($n_d - 1$). The torus topology tolerates more faults than the mesh due to the fact that packets can move in both directions of a dimension (in particular, $2n_d - 1$ faults). The fat–tree topology tolerates as many faults as the number of up or down ports of the switches minus one ($k_i - 1$). In the FB, each router is connected to $(k_f - 1)n_f$ routers, so it tolerates as many faults as $(k_f - 1)n_f - 1$.

In the case of the KNS family of topologies and considering faults in the links connected to routers, as long as a router is still connected to one dimension, it may forward packets, providing that the indirect subnet associated to

that dimension is working. Therefore, at least, the number of tolerated faults is given by the number of dimensions minus 1. If the faults occur in the links of the indirect network, as long as one subnet of every dimension is working, they will be also tolerated. Remember that there are k_h subnets per dimensions. Indeed, if fat-trees are used as the indirect subnets, several faults in each one of them are tolerated. Notice, though, that RUFT is not fault-tolerant, since there is a unique path for each source–destination pair, so it tolerates 0 faults. However, even when using RUFTs as indirect subnets, as long as other indirect subnets of the same dimension are working, the number of tolerated link faults in different indirect subnets should be higher than the one of the links connected to routers. As a consequence, we conclude that the fault tolerance degree of the new topology is upper bounded by the maximum number of faults in the routers, that is, the number of dimensions minus one, $n_h - 1$. This gives us the same fault tolerance as a mesh with the same number of dimensions.

Another analysis is also possible. Assume that we have routers with p ports available. With such a router, we could build a mesh with $\frac{p}{2}$ dimensions that tolerates $\frac{p}{2} - 1$ faults or a torus with also $\frac{p}{2}$ dimensions that tolerates $2\frac{p}{2} - 1 = p - 1$ faults. In the case of the KNS, we could built a p dimensional network that tolerates $p - 1$ faults. That is, for the same router degree, the KNS tolerates the same number of faults as a torus.

On the other hand, considering the rich connectivity of the newly proposed topology, a higher number of faults should be tolerated with a very high probability using a fault-tolerant routing algorithm or reconfiguration mechanism. Finally, it must be noticed that routing should be also changed to fully support fault tolerance in all topologies. However, an in depth analysis of both fault-tolerance probability and fault-tolerance routing issues is out of the scope of this paper.

We have not considered faults in injection links. Most topologies usually have a single link that connects the processing node to the network. If this link fails, the processing node will be isolated. Therefore, considering faults in the injection links, these networks will not tolerate any fault. However, in the KNS topology family (and also in tori and meshes), if the router is implemented inside each processing node (for instance using the HCA cards from [1]), the processing node is actually connected to the network through as many links as dimensions in the network, therefore tolerating also faults in the injection links.

6 Related Work

There are previous works that propose alternative topologies to the ones considered in this paper, but they have been never or seldom used in commercial products or in supercomputers. This is the case of the WK–recursive topology that was proposed in [13] for interconnection networks and more recently for on–chip networks [27], but, to the best of our knowledge, it has never been used

in commercial products. Moreover, this topology has difficulties to guarantee deadlock freedom in the routing algorithm.

One of the topologies considered for comparison purposes in this paper is the flattened–butterfly[22] which is a very popular topology in recent papers. It is obtained from combining the routers in each row of a conventional butterfly MIN, thus obtaining an n–dimensional direct network where the nodes of each dimension are not connected in a ring fashion like in a torus, or through a small indirect topology like in our proposal; instead they are fully connected. In this paper, we have referred to n_f as the number of network dimensions and to k_f as the number of switches per dimension. This results in a topology very similar to a generalized hypercube but attaching several processing nodes to the same switch. Therefore, the flattened–butterfly, like the generalized hypercube, has a high cost, specially for large machines, which are the focus of our proposal. This topology is compared against our proposal in Section 5.

Other works propose the combination of several topologies, as we do in this paper. Most of them have been proposed for on–chip networks and therefore the target is different to ours. For example, in [26], each core is connected to two different tree networks in an on-chip environment in order to overcome the poor performance provided by trees. This proposal is not suitable for large machines due to the complexity of the cable layout and the poor performance achieved even with two trees. Another proposal is the multi-ring topology [9], which is composed of several interconnected rings.

Many of the proposals for on–chip networks are based on hierarchical topologies. Subsets of cores are connected by small local networks connected in turn, by a global network. This is not the case of the family of topologies proposed in this paper. Hierarchical designs are expected to have a higher latency and smaller throughput, since both networks must be traversed for most of the source–destination pairs. In [12], the authors propose to use as local network a simple bus, and a mesh as global network. As can be expected, this is not an appropriate topology for a large machines due to the low performance provided by buses and meshes. The authors of [21] propose a tool to select the most suitable topology for a given network design. The tool explores the design space of hybrid Clos-torus networks. However, opposite to our proposal, the explored designs are hierarchical topologies, where local networks are Clos networks and they are connected by a global torus network. Another hierarchical topology is the DragonFly[23, 17], which provides a topology that is based on grouping routers virtual routers to increase the effective radix of the network. This topology uses two different networks, one intra–group, and another one inter–group. For this topology, it is advisable to use a non-minimal global adaptive routing to balance the load across the global channels. These global channels, which link different groups, are long links, so that a high latency can be expected.

A topology closer to the new family proposed in this paper is the mesh of trees (MoT) introduced in [24] and later used also in NoCs [7]. It is based on an $n-$dimensional topology where the nodes of a given dimension are connected using a tree. This results in a particular case of our proposed family

of topologies with very poor expected performance due to using a simple tree for connecting the nodes of a dimension. The Bcube [20] and Hypercrossbar network [10] also resemble our proposal. Each processing node is connected to several dimensions by using several NICs. A switch is used to connect the processing nodes of the same dimension. However, routing is performed through end-processing nodes, by ejecting packets from the network through a NIC and later reinjecting them through another one.

Finally, in [29] a topology that combines several tori networks is proposed. The proposal focus on large supercomputers, but its applicability is limited to 2^{16} processing nodes and its wiring layout is complex for large machines. The proposal starts from a 2-D torus and provides bypass links in the diagonal direction as many times as needed. This proposal does not improve the number of hops in a single dimension, since no new links are added to connect nodes of the same dimension, but reduces the number of hops when traversing several dimensions. In addition, this topology has the same problem with deadlock-free routing than the WK–recursive.

7 Conclusions

This paper proposes a new family of hybrid topologies, the KNS, for large–scale interconnection networks. It is based on an n–dimensional topology where the nodes of each dimension are connected through a crossbar or a small indirect topology (a fat–tree or a RUFT in this paper). This results in a new family of topologies that provides high–performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but with a much lower hardware cost. In particular, from the throughput point of view, the new topologies with fat–trees as indirect subnet are the best ones. Nevertheless, from the cost–performance point of view, the new topologies with crossbars as indirect subnets are the winners, as they are able to obtain better throughput per dollar compared to indirect topologies, and the differences are even higher when comparing to direct topologies. Moreover, in the new topologies with MIN's as indirect subnets, as the indirect subnets are small, the layout of the new topologies is much simpler than the one for indirect topologies with the same number of processing nodes. Concerning fault–tolerance, the proposed family of topologies is able to tolerate, at least, the same number of faults as a mesh with the same number of dimensions, regardless the topology used in the indirect subnets.

Acknowledgements This work was supported by the Spanish Ministerio de Economa y Competitividad (MINECO) and by FEDER funds under Grant TIN2012-38341-C04-01 and by Programa de Ayudas de Investigación y Desarrollo (PAID) from Universitat Politècnica de València.

References

1. Connect-IB. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_Connect-IB.pdf
2. Mellanox Store. http://www.mellanoxstore.com
3. Mellanox Technology. http://www.mellanox.com
4. Myricom. http://www.myri.com
5. Quadrics homepage. http://www.quadrics.com
6. TOP500 Supercomputer Site. http://www.top500.org
7. Balkan, A., Qu, G., Vishkin, U.: Mesh-of-Trees and Alternative Interconnection Networks for Single-Chip Parallelism. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **17**(10), 1419–1432 (2009). DOI 10.1109/TVLSI.2008.2003999
8. Bermudez Garzon, D., Gomez, M.E., Lopez, P., Duato, J., Gomez, C.: FT-RUFT: A Performance and Fault-Tolerant Efficient Indirect Topology. In: Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, pp. 405–409. IEEE (2014)
9. Bhandarkar, S.M., Arabnia, H.R.: The Hough Transform on a Reconfigurable Multi-Ring Network. J. Parallel Distrib. Comput. **24**(1), 107–114 (1995)
10. Boku, T., Nakazawa, K., Nakamura, H., Sone, T., Mishima, T., Itakura, K.: Adaptive routing technique on hypercrossbar network and its evaluation. Systems and Computers in Japan **27**(4), 55–64 (1996)
11. Dally, W., Towles, B.: Principles and practices of interconnection networks. Morgan Kaufmann (2004)
12. Das, R., Eachempati, S., Mishra, A., Narayanan, V., Das, C.: Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In: High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on, pp. 175–186 (2009). DOI 10.1109/HPCA.2009.4798252
13. Della Vecchia, G., Sanges, C.: Recursively Scalable Networks for Message Passing Architectures. Proceedings of International Conference on Parallel Processing and Applications pp. 33–10 (1987)
14. Duato, J.: A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks. IEEE Transactions on Parallel and Distributed Systems **7**, 841–854 (1996). DOI 10.1109/71.532115
15. Duato, J., Yalamanchili, S., Lionel, N.: Interconnection Networks: An Engineering Approach. Morgan Kaufmann Publishers Inc., USA (2002)
16. Flich, J., Malumbres, M., López, P., Duato, J.: Improving Routing Performance in Myrinet Networks. Parallel and Distributed Processing Symposium, International p. 27 (2000). DOI 10.1109/IPDPS.2000.845961
17. García, M., 0001, E.V., Beivide, R., Camarero, C., Valero, M., Rodríguez, G., Minkenberg, C.: On-the-fly adaptive routing for dragonfly interconnection networks. The Journal of Supercomputing **71**(3), 1116–1142 (2015)
18. Gómez, C., Gilabert, F., Gómez, M., López, P., Duato, J.: Deterministic versus Adaptive Routing in Fat-Trees. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp. 1–8 (2007). DOI 10.1109/IPDPS.2007.370482
19. Gómez, C., Gilabert, F., Gómez, M., López, P., Duato, J.: RUFT: Simplifying the Fat-Tree Topology. In: Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on, pp. 153–160 (2008). DOI 10.1109/ICPADS.2008.44
20. Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., Lu, S.: BCube: a high performance, server-centric network architecture for modular data centers. In: SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, pp. 63–74. ACM, New York, NY, USA (2009). DOI 10.1145/1592568.1592577. URL http://www.bibsonomy.org/bibtex/23a5da89fbf099e3c70f4559ab38082c5/chesteve
21. Gupta, A., Dally, W.: Topology optimization of interconnection networks. Computer Architecture Letters **5**(1), 10–13 (2006). DOI 10.1109/L-CA.2006.8
22. Kim, J., Dally, W., Abts, D.: Flattened butterfly: a cost-efficient topology for high-radix networks. In: Proceedings of the 34th annual international symposium on Computer architecture, ISCA '07, pp. 126–137. ACM, New York, NY, USA (2007). DOI 10.1145/1250662.1250679

23. Kim, J., Dally, W., Scott, S., Abts, D.: Technology-Driven, Highly-Scalable Dragonfly Topology. In: Proceedings of the 35th Annual International Symposium on Computer Architecture, ISCA '08, pp. 77–88. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/ISCA.2008.19
24. Leighton, F.: Introduction to parallel algorithms and architectures: arrays, trees, hypercubes. v. 1. M. Kaufmann Publishers (1992)
25. Leiserson, C.E.: Fat-trees: universal networks for hardware-efficient supercomputing. IEEE Trans. Comput. **34**(10), 892–901 (1985)
26. Matsutani, H., Koibuchi, M., Amano, H.: Performance, Cost, and Energy Evaluation of Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp. 1–10 (2007). DOI 10.1109/IPDPS.2007.370271
27. Rahmati, D., Kiasari, A., Hessabi, S., Sarbazi-Azad, H.: A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips. In: Computer Design, 2006. ICCD 2006. International Conference on, pp. 142–147 (2006). DOI 10.1109/ICCD.2006.4380807
28. Towles, B., Dally, W.J.: Worst-case traffic for oblivious routing functions. In: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, SPAA '02, pp. 1–8. ACM, New York, NY, USA (2002). DOI 10.1145/564870.564872. URL http://doi.acm.org/10.1145/564870.564872
29. Yang, Y., Funahashi, A., Jouraku, A., Nishi, H., Amano, H., Sueyoshi, T.: Recursive diagonal torus: an interconnection network for massively parallel computers. Parallel and Distributed Systems, IEEE Transactions on **12**(7), 701–715 (2001). DOI 10.1109/71.940745