A case study on implementing virtual 5D torus networks using network components of lower dimensionality

Francisco J. Andújar[†], Juan A. Villar[*], José L. Sánchez[*], Francisco J. Alfaro[*],
José Duato[†] and Holger Fröning[‡]

[*]Department of Computing Systems, University of Castilla-La Mancha, Albacete, Spain
Email: {juanan, jsanchez, falfaro}@dsi.uclm.es
[†]Department of Systems Data Processing and Computers, Polytechnic University of Valencia, Valencia, Spain
Email: fandujarm@gap.upv.es, jduato@disca.upv.es
[‡]Institute of Computer Engineering, Ruprecht-Karls University of Heidelberg, Mannheim, Germany
Email: holger.froening@ziti.uni-heidelberg.de

***Abstract*—Several of the most powerful supercomputers in the Top500 and the Graph500 lists continue choosing a torus topology to interconnect a large number of compute nodes. In some cases, a torus network with five or six dimensions is implemented, however, one notices that the costs of implementing an interconnection network increase with the node degree. In previous works we defined and characterized the nD Twin (nDT) torus topology in order to virtually increase the dimensionality of a torus. This new topology reduces the distances between nodes and therefore increases network performance. In this work, we present how to build a 5DT torus network using commercial 6-port network cards. The main issues of this approach are detailed, and we present solutions these problems. Moreover we show, using the same components, that the performance of the 5DT torus network is higher than the performance of the 3D torus network for the same number of compute nodes.**

I. Introduction

Nowadays, high performance computing (HPC) systems can comprise thousands of nodes. This large amount of nodes imposes severe performance requirements on the interconnection network, which plays a major role in determining the overall system performance.

There are many design issues that may affect the choice of an appropriate interconnection network. Among them, the network topology has a significant impact on the interconnection network performance in this kind of systems, and therefore its choice is an important design decision. Fat-tree [1] and torus topologies [2] are widely used for implementing indirect and direct networks, respectively. In November 2016, there are four supercomputers using the torus topology in the top ten of the Top500 list [3] and seven in the top ten of the Graph500 list [4] (e.g., K-Computer [5] and several supercomputers of Blue Gene/Q family [6]), while the remaining supercomputers use some kind of indirect network topologies (e.g., fat-tree or Dragonfly).

When a HPC system grows, there are important network characteristics to consider in addition to the network performance, such as economical cost, power consumption, reliability or scalability. Although the fat-tree topology provides equal access bandwidth to every node and is appropriate for running parallel applications that generate a lot of communication among all the nodes, the torus topology provides a reduced hardware and an excellent scalability, allowing an easier implementation when the number of nodes grows. In addition, the torus topology supports several routing algorithms that increase the path diversity so that the fault tolerance and load balance become feasible. For all these reasons, the torus topology is a common topology used in the greatest HPC systems, according to the Top500 and the Graph500 lists.

In torus topologies, the network performance is very dependant on the number of dimensions. When the number of dimensions is increased, the average number of hops between any pair of nodes is reduced and therefore, the network latency is also reduced. Then, the higher the number of dimensions, the higher the network performance obtained. However, to build torus networks with more dimensions requires to increase the number of ports in the communication hardware, increasing its complexity. Therefore, the network deployment becomes more expensive; and once the HPC system is running, the maintenance cost is also higher.

In previous works, we have proposed a new topology, called n-dimensional twin torus, or just nDT torus [7], which allows us to increase the number of dimensions of the torus. For example, if we have 4-port communication cards to build a torus network, we can build a 2-dimensional (2D) torus[1] or we can build a 3DT torus [8]. In the 3DT torus, each node in the network comprises two 4-port cards: one port of each card is used to interconnect the cards and the six remaining ports are used to connect the node to its neighbours in the three dimensions of the 3D torus.

[1]Note that two ports are required for each dimension in the n-dimensional torus.

In general, an nDT torus can be built using $(n+1)$-port cards: one port from each card interconnects both cards and the 2n remaining ports compose the nDT torus node. In [7], we show how the network diameter and average distance decrease when building an nDT torus instead of a $\frac{n+1}{2}D$ torus. As a consequence, the performance of the network is increased without extra economic investment.

In the previous work, we evaluated the nDT torus topology by simulation, assuming a simple card architecture to model the network. In this paper, we present a new study using a more accurate model based on commercial communication hardware. Specifically, we have developed a model based on the EXTOLL technology [9], [10]. EXTOLL permits to contruct direct networks with a node degree of six, and was designed specifically for the use in high-performance computing. It comes with dedicated support for fine-grained communication, as well as bulk data transfers. It features state-of-the-art techniques like virtual output queueing, virtual channels and link-level retransmission. Since EXTOLL cards have six communication ports, the common approach is to build a 3D torus network, but using our proposal, to build a 5DT torus network is feasible.

Due to the specific characteristics of EXTOLL cards, we have had to incorporate a new crossbar architecture into our previous network model, including a new flow control mechanism, a new routing algorithm, among other resources that will be outlined in Section III. The rest of the paper is organized as follows: In Section II we present an overview of the EXTOLL architecture and the nDT torus topology. Section III explains the implementation of the EXTOLL model and the implementation of the 5DT torus using the EXTOLL model. In Section IV, we evaluate by simulation the performance of several 3D torus and 5DT torus networks with the same number of nodes. Finally, Section V outlines the conclusions.

II. Background

In this section we review the previous work. Section II-A shows a brief description of the EXTOLL card and a more detailed description of the EXTOLL switch, which is the focus of this paper. Section II-B formally defines the nDT torus and shows a brief explanation about the nDT torus port configuration and the nDT torus routing algorithm.

A. The EXTOLL technology

EXTOLL is an interconnection network technology designed for achieving a very low latency, a high bandwidth, a high sustained message rate and a high availability in large-scale networks (up to 64k nodes). EXTOLL cards offer six ports, allowing to build any direct topology with a maximum degree of six. Among them, the 3D torus is the preferred topology for large networks.

Figure 1 shows the top-level block model of the EXTOLL architecture. Three main logical blocks can be distinguished: the host interface, the network interface controller or NIC and the network switch, drawn in green, blue and red, respectively.

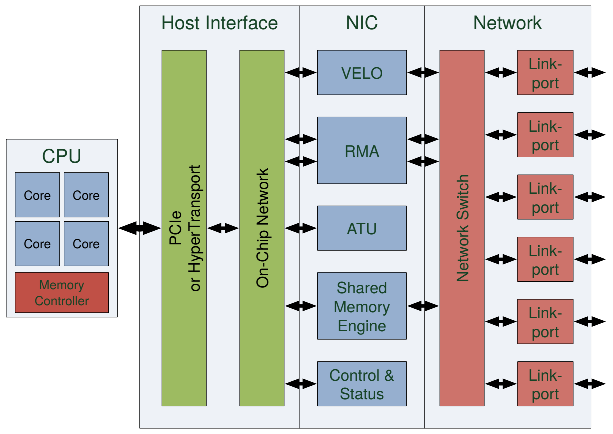


Fig. 1. Top-level block diagram of EXTOLL architecture.

The host interface connects EXTOLL to the host system using PCIe or HyperTransport interface.

The second block, the EXTOLL NIC, implements different modules to transform the packets sent through the host interface to network packets and vice versa. There are five functional units (FUs) integrated into the EXTOLL NIC. The Virtualized Engine for Low Overhead (VELO) [11] unit is dedicated to low latency transfer of small messages, while the Remote Memory Access (RMA) [12] unit employs Direct Memory Access to handle large messages. The Address Translation unit (ATU) is used to fast translation from virtual address to physical address for the RMA. The Shared Memory Functional Unit (SMFU) [13] allows load/store forwarding between the different address spaces of multiple nodes. Finally, the Control & Status Register unit is used to configure the EXTOLL card, to acquire status information and for debugging purposes.

The third block implements a complete crossbar-based switch. The main EXTOLL switch features are described below, although there are other features not discussed here, such as the multicast hardware support, since these characteristics are out of scope of this work.

The EXTOLL switch has ten ports: four of them connect the switch with the NIC FUs, while the remaining six ports interconnect the EXTOLL switch with other switches. The network switch implements virtual cut-through [14] as switching technique, the iSLIP scheduling algorithm [15] and a credit-based flow control.

The EXTOLL switch is an IQ (Input Queued) switch [16], [17]; i.e., there are only buffers at the input ports. A typical problem of IQ switches is head-of-line (HOL) blocking. Basically, when the first packet in the buffer is blocked, the remaining packets stored in the buffer and destined to other output ports which could be forwarded, are also blocked. Since the HOL blocking can reduce dramatically the network performance, EXTOLL implements virtual output queuing (VOQ) [14], [18] at switch level; i.e, each packet is stored in a different queue depending on its output channel, and therefore, the negative effect of the HOL blocking is reduced.

To provide a quality-of-service mechanism, the EXTOLL switch offers four different Traffic Classes (TCs). The software decides on which TC the packets are injected, and the packets cannot change their TC during their transmission. Each TC has two deterministic virtual channels (VCs) to enable the implementation of deadlock-free routing algorithms [19]. Moreover, each TC can perform its own routing function in order to balance the network load; and there are two TCs that include a third VC to allow the implementation of adaptive routing algorithms [20]. In order to support multiple TCs, VCs and the VOQ-switch technique, the EXTOLL switch includes multi-queue FIFO buffers [21].

The EXTOLL switch implements a table-based routing algorithm. Each input port has a dedicated routing table, thus allowing that multiple input ports perform the routing simultaneously, and it is not necessary to arbitrate among multiple input ports trying to read the same routing table. Moreover, the table-based routing allows to build arbitrary topologies and to recover the network when there are component failures. The main drawback of the table-based routing is that large tables are required at each input port. EXTOLL implements hierarchical routing tables, reducing significantly the size of the required tables.

Finally, note that the packets in EXTOLL switches have a variable size. The minimum data granularity in the EXTOLL switch is the cell, a data chunk of 128 bits. The packet size can vary from 1 cell to 32 cells. The variable packet size allows to perform large data transfers more efficiently and does not waste buffer resources for small data transfers.

Furthermore, in order to improve the buffer usage, a fine grain credit flow control is implemented. If each packet would consume one credit, each credit consumption would have to block the maximum packet size in order to not overflow the buffer, independently of the real size of the packet. This would be very inefficient since if there are a lot of small packets travelling in the network, the credits could be consumed although most of the buffer space was free. For this reason, each packet is logically split in multiple smaller parts (cells), consuming one credit for each part. Therefore, a packet consumes one or several credits. This scheme allows to use the buffers more efficiently.

B. The nDT torus

In order for this paper to be self-contained, we include a minimal set of definitions and a brief explanation regarding the nDT torus. More details can be found in [7], [8]. First, we introduce the notation used and formally define the nD twin torus topology [7].

Notation

- n: number of dimensions of the torus, where $n \geq 2$.
- d_i: a dimension of the nD torus (or the nDT torus), $0 \leq i < n$.
- d_i^+, d_i^-: ports of the dimension d_i.
- PE0, PE1: processing elements of a nDT torus node.

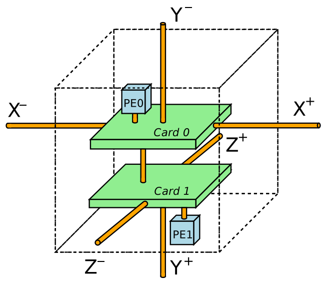
Definition 2.1: An nD Twin torus, or just nDT torus, is an n-cube k-ary (nD torus) topology, with $k \in \mathbb{N}^*$, $k \geq 2$ and $n \geq 3$, where each node is basically composed of the following main components:

- Communication hardware: it consists of two $(n+1)$-port cards[2], offering a total of $2n+2$ ports. Two of these ports (one in each card) are used to interconnect the cards, and the $2n$ remaining ports are used to connect the node to the other dimensions, building a torus topology with n dimensions.
- Computing hardware: each internal $(n+1)$-port card is connected to a processing element (PE), and so there are two PEs in each node. Therefore, there is a total of $2k^n$ PEs interconnected by the network.

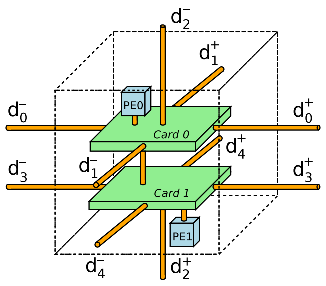
For example, the 3DT torus and the 5DT torus nodes, shown in Figure 2, comprise two 4-port cards and two 6-port cards, respectively. Both nodes have two PEs[3].

Then, each node has $2n$ ports split in two cards. To establish communication between ports belonging to different cards, it is necessary to use the internal link[4], increasing the communication latency. Hence, the important issue for nDT torus topology is that, in order to reduce the latency, it is important to avoid, as much as possible, the paths (and therefore, the messages) passing through the internal link.

We performed an analytical study in order to determine the optimal node configuration for the nDT torus [7]. We considered for the study the DOR (Dimension Order Routing)



(a) 3DT torus node (4-port cards).



(b) 5DT torus node (6-port cards).

Fig. 2. Examples of nDT torus nodes.

[2]When only one $(n+1)$-port card is used per node, an n_0D torus is obtained, being $n_0 = \frac{n+1}{2}$.

[3]The processing elements of current supercomputers are usually integrated on the same board with several cores, caches, memory, etc. However, from the topological point of view, we only need to consider the fact that packets are sourced/destined from/to the PEs and the particular internal layout of PEs is not relevant.

[4]Henceforth, we will use "internal link" to refer to the connection between the two cards in an nDT torus node, and "external links" to refer to the remaining ports.

routing algorithm [2]. This deterministic routing algorithm is commonly used in k-ary n-cubes because it is a very simple routing algorithm. Moreover, DOR is deadlock-free using two virtual channels [19] or combining it with the bubble flow control mechanism [22]. After selecting the routing algorithm, we have determined the optimal configuration of the nDT torus node, shown in Definition 2.2. Figure 2 shows the optimal node configuration for the 3DT torus and the 5DT torus.

Definition 2.2: Given two communication cards, each one with $(n+1)$ ports, being n an odd number[5] greater or equal than 3, the port configuration that minimizes the number of paths crossing the internal link in an nDT torus node is defined as follows:

- The ports belonging to dimensions from d_0 to $d_{\frac{n-1}{2}-1}$ are connected to Card0.
- The ports belonging to dimensions from $d_{\frac{n-1}{2}+1}$ to d_{n-1} are connected to Card1.
- The two ports of the dimension $d_{\frac{n-1}{2}}$ are distributed between the two cards. Since k is odd, the number of paths that cross the internal link is the same, regardless of the card in which each port is connected. Hereon in, we assume that the port $d^-_{\frac{n-1}{2}}$ is connected to Card0 and the port $d^+_{\frac{n-1}{2}}$ is connected to Card1.

Once the optimal configuration is obtained, a few modifications to the DOR algorithm are required in order to route the messages in the nDT torus. We call this modified version the DORT routing algorithm. Basically, a message is routed by the n dimensions following a strictly ascending (or descending) order. Since the nDT torus node comprises two internal cards, once the output port is selected, DORT checks whether the output port is connected to the current card. Depending on the result, the message is routed to the output port or to the internal link. Finally, when the message reaches the destination node, destination PE is checked. The message is routed to the NIC if the destination is the PE attached to the current card, or it is routed to the internal link if the destination is the other PE in the node.

However, this routing algorithm is not deadlock-free due to the use of the internal link. Fortunately, this problem can be solved by adding a few virtual channels to the internal link (0, 1 or at most 2 extra virtual channels, depending on the number of dimensions and the mechanism chosen to avoid the deadlock). No modification in external links is required. A more detailed analysis is shown in [7].

Since this study is focused in building 5DT torus networks using EXTOLL cards, and EXTOLL cards use virtual channels to avoid deadlock, we show a brief explanation about how DORT routing ensures the deadlock-freedom in 5DT tori using virtual channels (VCs).

If we analyse in detail the use of the internal link, we can distinguish three cases, depending on the destination of the message after using the internal link: i) The message destination is the PE connected to the other card in the 5DT torus node[6]; ii) the message uses the internal link to be injected into the network in a dimension connected to the other card (e.g., a message is routed in Card0 to d_4); iii) the message is travelling through dimension d_2.

Then, a message can use the internal link regardless of the dimension where it is travelling. New cycles could be generated and as a consequence, the deadlock could appear. For example, let's consider a message routed in a 5DT torus, such that the message needs to use the five dimensions following an ascending order. First, the message is routed in d_0, after that the message is routed in d_1 and so on. We know that when the message has already been routed in d_1, the message cannot be routed in d_0 again, avoiding the cycles between messages travelling in d_0 and d_1. However, the use of an internal link does not follow any order. The message is routed to the internal link before travelling in d_0, after travelling in d_4, travelling through d_2, etc.

To avoid that, the internal link requires four VCs instead of the two VCs required by the external links. The first and second VCs are used for sending the messages of the case i) and ii), respectively. The third and the fourth VCs are used for sending the messages of case iii). Since the message is routed in d_2, two VCs are required for case iii), in the same way that the external links require two VCs.

III. Modelling EXTOLL Cards

As mentioned above, the purpose of this work is to address a study of the 5DT torus topology built using the EXTOLL switch. Obviously, the most accurate study would be obtained using a real cluster whose interconnection network is composed of hundreds of EXTOLL cards. However, this option is unfeasible because we do not have access to a cluster of these characteristics. Moreover, if available, a lot of time is needed to reconfigure the hardware to perform the experiments, and most important, the system reconfiguration could interfere with the work of other cluster users. For these reasons, as usual, we have performed this study by simulation.

Then, the first step is to develop an EXTOLL switch model that is accurate enough. We refer to this model as the EXTOLLsim model. Since we want to evaluate the network performance from the topological point of view, we have focused on the development of a model for simulating the EXTOLL crossbar. The host interface is not modelled and the model of the NIC FUs has been simplified. In other case, the development cost would have been higher and the obtained results would not be more significant than using a simpler model. For example, if the host interface would have been included in the EXTOLLsim model, there would be a latency in the communication introduced by the host interface model. However, this latency would be independent of the network topology and would have no effect for the purposes of this

[5]The nDT torus node for even values of n is also defined in [7], but we have omitted this definition here because the study presented in this paper mainly focuses on 3DT and 5DT tori.

[6]Remember that the ports of dimensions d_0, d_1 and the port d_2^- are connected to Card0, while the ports of dimensions d_3, d_4 and the port d_2^+ are connected to Card1, as can be observed in Figure 2b.

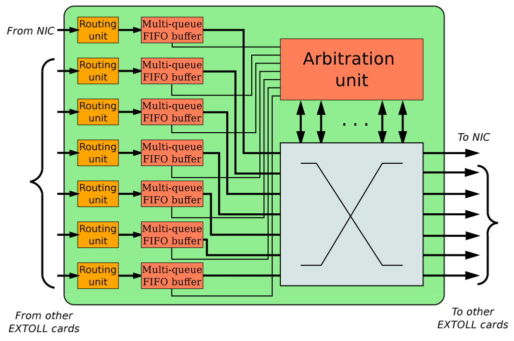


Fig. 3. EXTOLLsim logical model.

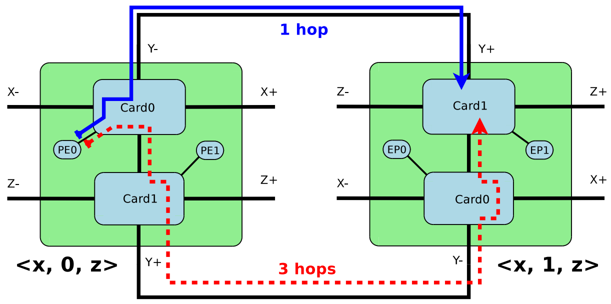


Fig. 4. Wrong (dashed line) and correct (solid line) Y link selection in case of tie routing in the Y dimension.

study. Therefore, the only significant result obtained would be to make the simulations more inefficient in terms of time.

Regarding the EXTOLL NIC, it has 4 FUs injecting traffic to the EXTOLL network. However, without a detailed traffic model, modelling all the FUs would only increase the memory consumption and the execution time of the simulations without significant differences in the results. For this reason, only one generic FU has been modelled without losing accuracy. This FU is not based on an specific FU of EXTOLL NIC, its only purpose is to generate and inject traffic into the network using synthetic traffic patterns or trace-based traffic. Therefore, the EXTOLLsim model has only seven ports instead of ten ports: one port is connected to the EXTOLL NIC and six ports are connected to other EXTOLL cards.

Regarding the EXTOLL network, we have developed a detailed model of the EXTOLL crossbar. The EXTOLLsim model comprises four logical units, shown in Figure 3: the routing units, the buffers, the arbitration unit and the crossbar unit. Each logical unit models different features of the EXTOLL crossbar and has its own latency.

When a packet arrives at an EXTOLLsim card from a given port, it is processed by the routing unit of that port[7]. Once the routing process finishes, the packet is stored in the buffer, which models a multi-queue FIFO buffer with VOQ-switch to reduce the negative effect of HOL blocking.

If there are packets stored in the buffers, the arbitration unit, which models the iSLIP scheduling algorithm [15], decides which packets will go through the crossbar unit. Note that the crossbar unit is not only responsible for forwarding the packets to the next EXTOLLsim card, it is also responsible for the proper function of the virtual cut-through switching technique and the fine-grain credit flow-control. Finally, as the real EXTOLL switch, the EXTOLLsim model also implements variable-sized packets, four traffic classes (TCs) with their multiples VCs, and supports multiple topologies and routing algorithms, including adaptive routing algorithms.

[7]Although the EXTOLL switch implements a table-based routing, the EXTOLLsim model implements the routing as hardware routing units. The reproduced behavior is the same but our implementation saves a big amount of main memory during the simulation because the routing tables are not modelled.

A. Deadlock-free routing for 5DT tori using the EXTOLL cards.

Once the EXTOLLsim model has been developed, the next step is to build the nD and nDT torus and to define the routing algorithm of both topologies. Since EXTOLL cards have six communication ports, a 3D torus and a 5DT torus can be built.

Regarding the routing algorithm, the DOR routing algorithm has been chosen for the 3D torus. As commented above, the DOR algorithm avoids deadlock using two VCs [19]. The EXTOLL crossbar provides two deterministic VCs per each TC in order to avoid deadlock. Then, when a packet is travelling in a dimension d_i, the d_i-coordinate of the current node and the d_i-coordinate of the destination node are compared. If the destination node d_i-coordinate is greater than the current node d_i-coordinate, the packet is routed to the first VC, called upper-VC. In other case, the packet is routed to the second VC, called lower-VC.

A third VC is used in the adaptive TCs allowing the implementation of a fully-adaptive routing algorithm for the 3D torus using the Duato's protocol [20]. A round-robin arbiter is used to choose among the selectable adaptive channels.

Nevertheless, the implementation of the DORT routing algorithm in the 5DT is not possible using EXTOLL cards. As commented in Section II-B, DORT requires four VCs in the internal link[8] to avoid deadlocks [7]. EXTOLL cards do not have enough deterministic VCs to implement this algorithm. However, an EXTOLL card still has the TCs. Each TC can use different paths to route packets destined to the same node. Then, we have designed a new routing algorithm for the nDT torus, based on the DOR routing algorithm, that avoids deadlock using the TCs. We call this algorithm TS-DOR (Twin Source Dimension Order Routing).

Many of the cycles introduced by the internal link are generated by the packets injected from the PE1. Using DORT, in most of the cases the packets generated by the PE1 must cross the internal link to be routed in d_0. However, this internal-link hop can be avoided if the packets are routed first in the dimensions connected to the Card1. There are some routing approaches in the field of network-on-chips that, in order to balance the network traffic, propose to use two VCs to route the packets following different dimension orders

[8]Remember that no modifications are required in the external links.

[23]. For instance, for 2D mesh topologies, in the first virtual network the packets are routed following $X - Y$ order, while in the second virtual network the packets are routed following $Y - X$ order.

TS-DOR employs a similar approach to avoid the unnecessary use of the internal link for the PE1-sourced packets: the PE0-sourced packets are routed from d_0 to d_{n-1} while the PE1-sourced packets are routed from d_{n-1} to d_0. In the EXTOLLsim model, we can use the TCs to avoid the deadlock between PE0-sourced packets (routed from d_0 to d_4) and PE1-sourced packets (routed from d_4 to d_0). Since there are four TCs available, two of them (one adaptive TC and one deterministic TC) are used to inject PE0-sourced packets and the remaining two are used to inject PE1-sourced packets.

However, this approach only avoids the deadlock between PE0-sourced packets and PE1-sourced packets. Fortunately, the two deterministic VCs of each TC are enough to avoid deadlock, employing the same approach used for the external links.

Therefore, when a packet is routed in dimension $d_{\frac{n-1}{2}}$ and uses the internal link, TS-DOR takes the same decision that would be chosen in the external links. If the destination node $(d_{\frac{n-1}{2}})$-coordinate is greater than the current node $(d_{\frac{n-1}{2}})$-coordinate, the message is routed to the upper-VC of the internal link; otherwise it is routed to the internal link lower-VC. If a packet is routed to the internal link for another reason, the upper-VC is chosen if the packet goes from Card0 to Card1; if the packet goes from Card1 to Card0 the lower-VC is chosen.

Finally, when the number of nodes in dimension $d_{\frac{n-1}{2}}$ is even, there are nodes equidistant from the positive and the negative $(d_{\frac{n-1}{2}})$-dimension ports. Taking a wrong decision while choosing the port causes the unnecessary use of the internal link. Let's consider a PE0-sourced packet in a 3DT torus with 2 nodes in the Y dimension. Commonly, after travelling through dimension Y, the packet will travel through dimension Z. Choosing the port Y^+, the packet needs three hops to arrive at Card1 of the other node, while choosing the port Y^-, the packet only needs one hop. Figure 4 shows the correct and the wrong decisions in this scenario. Then, to avoid the unnecessary use of the internal link in case of tie, the port $d^-_{\frac{n-1}{2}}$ is chosen for PE0-sourced packets, while the port $d^+_{\frac{n-1}{2}}$ is chosen for PE1-sourced packets.

IV. Performance evaluation

In this section, we compare by simulation the 3D torus and the 5DT torus performance using the EXTOLLsim model. **The main objective of this evaluation is to show that, using the same number of EXTOLL cards to build the 3D torus, we can improve the network performance building a 5DT torus.** Note that, unlike previous studies, we do not evaluate the 5D torus. Since EXTOLL cards have 6 ports, building a 5D torus is not possible and performing this evaluation makes no sense.

Specifically, we evaluate 3D and 5DT tori with 256, 512, and 1024 PEs. Table I shows the topologies for each network size[9]. Remember that we consider PEs, not cores. Each PE can have several cores, but has only one NIC. From this study viewpoint, if the 256-PE network has 256 cores (using single-core nodes) or 4096 cores (using 16-core nodes) does not matter, since the number of NICs and switches are the same in both cases. Although the evaluated networks are small compared with the networks of the greatest Top500-list supercomputers, these network sizes are reasonable for building supercomputers in research centres with limited economic resources.

TABLE I
3D and 5DT torus evaluated.

Number of PEs	Topology	
	3D torus	5DT torus
256	8×8×4	4×4×2×2×2
512	8×8×8	4×4×2×4×2
1024	16×8×8	4×4×2×4×4

Regarding the network workload, we consider an scenario with synthetic traffic model. First, we describe the different case studies used in the simulations. After that, we provide the simulation results and analyse them.

The network performance is evaluated generating the workload synthetically. We consider uniform traffic pattern for modelling the destination distribution because it is commonly used in network performance evaluations [2]. Regarding the packet size, its values are uniformly distributed from the minimum packet size (1 cell) to the maximum packet size (32 cells).

We have performed a set of tests, varying the topology in each case. Each test consists of 30 different experiments, obtaining the normalized average throughput and the average cell latency of each test case[10]. Note that the normalized injection rate and the normalized average throughput are the percentage of the host bandwidth injected and received per each NIC, respectively[11].

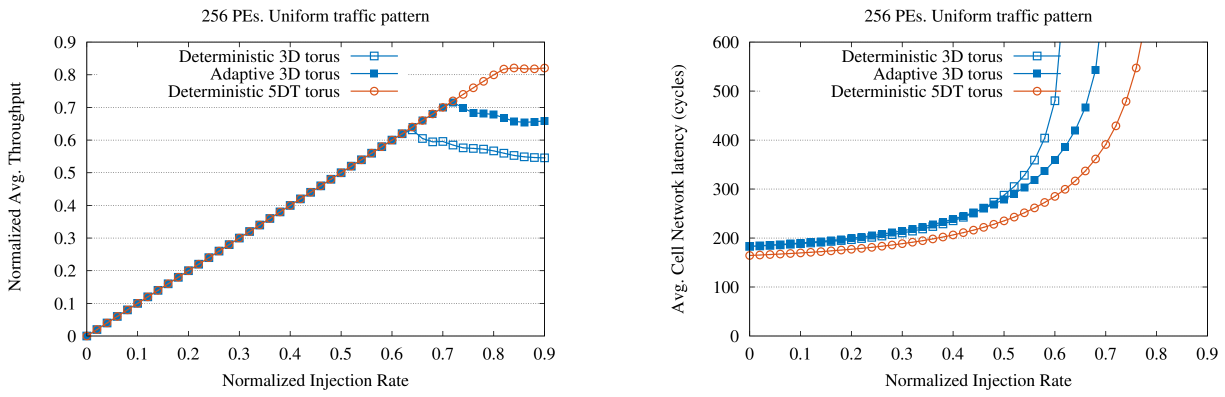
Figures 5, 6 and 7 show the results obtained for the 256-PE tori, the 512-PE tori and the 1024-PE tori, respectively. As seen, the 5DT torus obtains a better performance than the corresponding 3D torus. When the network is not saturated, the 5DT torus reduces the average network latency 10%, 15% and 30% for the 256-PE, the 512-PE and the 1024-PE tori, respectively, regardless the traffic pattern used.

Moreover, the 5DT torus saturates later, increasing the accepted traffic. For 256-PE and 512-PE tori, the 5DT torus achieves 20% and 12% more throughput than the corresponding 3D torus. But the increment of performance is more dramatic in the 1024-PE torus: the 5DT torus achieves 80% of the maximum throughput, while the corresponding 3D

[9]Note that 5DT and 3D networks have the same number of PEs since the 5DT torus nodes comprise two PEs.

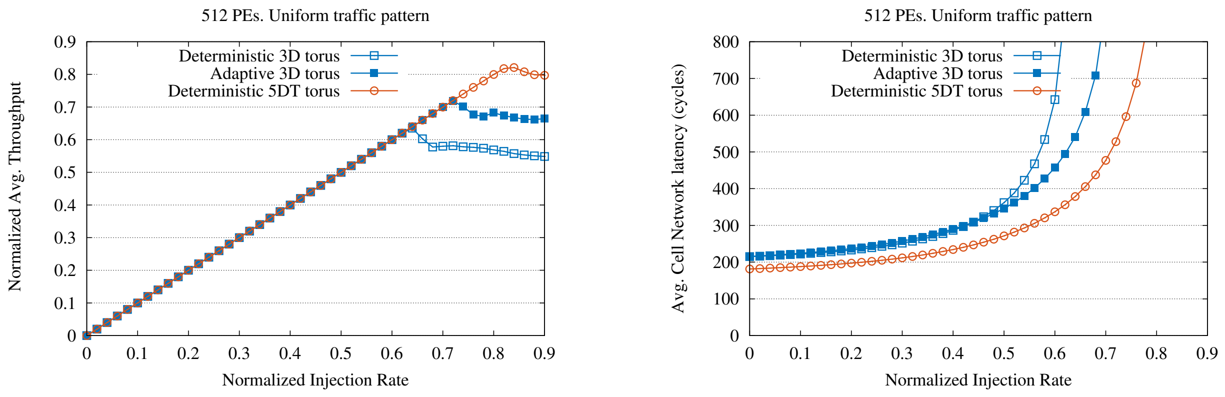
[10]We have also computed the confidence interval at 95%, but these intervals are imperceptible in the charts at first glance and we preferred not to include them.

[11]Since the host bandwidth is 10.4 GBytes/s, each 10% of the normalized injection rate is approximately 1 GByte/s.



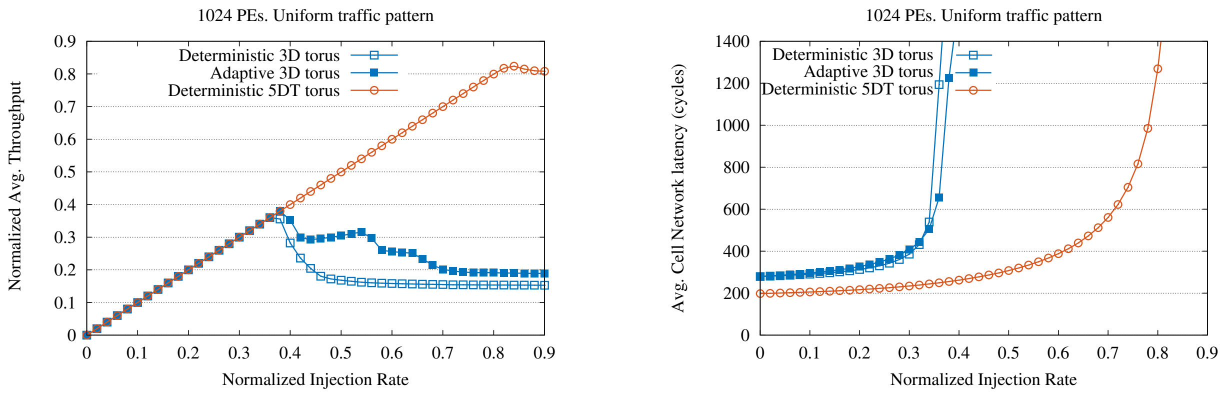
(a) Uniform traffic pattern.

Fig. 5. Network performance using synthetic traffic patterns for 256-PE tori.



(a) Uniform traffic pattern.

Fig. 6. Network performance using synthetic traffic patterns for 512-PE tori.



(a) Uniform traffic pattern.

Fig. 7. Network performance using synthetic traffic patterns for 1024-PE tori.

torus only achieves 35%. This performance degradation is common in large torus networks that employ virtual channels for avoiding deadlock [24], [25]. Since the number of nodes per dimensions in the 1024-PE 5DT torus is low (a maximum of 4 nodes per dimension against the maximum of 16 nodes in the corresponding 3D torus), the 5DT torus is not affected by this degradation.

Note that the 3D torus performance is improved using an adaptive routing. In any case, the 3D torus network latency is still higher when the network is not saturated.

V. Conclusions

In previous works, we showed how to build an nDT torus combining two cards of $(n+1)$ ports and how the network performance is increased building the nDT torus instead of the mD ($m = \frac{n+1}{2}$) torus. Building an nDT torus the distances between network nodes are reduced and therefore the performance is increased.

Now, we apply those previous works to implement an nDT torus using EXTOLL high performance interconnection cards to perform the study. They have six ports allowing to build a 3D torus and also support arbitrary topologies, allowing to build an nDT torus topology, or more specifically, a 5DT torus topology. For this reason, we include the EXTOLL switch architecture in our previous network model. The implementation of the EXTOLL model is also discussed.

For the nDT torus we have proposed the DORT routing algorithm to minimize the use of the internal link. Unfortunately, DORT cannot be applied to nDT tori since EXTOLL switches do not have enough deterministic virtual channels. For this reason, we develop a new deterministic deadlock-free routing algorithm called TS-DOR (Twin Source DOR) that combines the use of EXTOLL VCs and EXTOLL TCs to ensure deadlock-freedom in a 5DT torus.

Finally, we compare the performance of the 3D torus and the 5DT torus using the EXTOLL simulation model. Both tori are evaluated under uniform traffic pattern. As expected, the 5DT torus increases the network performance without changing the switch architecture, only modifying the topology and the routing algorithm.

Acknowledgment

This work has been supported by the Spanish MINECO and European Commission (FEDER funds) under the projects TIN2015-66972-C5-1-R and TIN2015-66972-C5-2-R; and by the JCCM under the project PEII-2014-028-P. Francisco J. Andujar is also funded by the Spanish MICINN under a Juan de la Cierva grant FJCI-2015-26080.

References

[1] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 892–901, 1985.

[2] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection networks. An engineering approach*. Morgan Kaufmann Publishers Inc., 2003.

[3] J. Dongarra, H. W. Meuer, and E. Strohmaier, "TOP500 Supercomputer Sites," June 2016, www.top500.org.

[4] "The Graph500 list," June 2016, http://www.graph500.org/.

[5] M. Yokokawa, F. Shoji, A. Uno, M. Kurokawa, and T. Watanabe, "The K-Computer: Japanese next-generation supercomputer development project," in *2011 International Symposium on Low Power Electronics and Design (ISLPED)*, 2011, pp. 371–372.

[6] D. Chen *et al.*, "The IBM Blue Gene/Q interconnection network and message unit," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–10.

[7] F. J. Andújar, J. A. Villar, J. L. Sánchez, F. J. Alfaro, and J. Duato, "N-dimensional twin torus topology," *IEEE Transactions on Computers*, vol. 64, no. 10, pp. 2847–2861, Oct 2015.

[8] F. J. Andújar, J. A. Villar, F. J. Alfaro, J. L. Sánchez, and J. Duato, "Building 3D torus using low-profile expansion cards," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2701–2715, Nov 2014.

[9] H. Fröning, M. Nüssle, H. Litz, C. Leber, and U. Brüning, "On achieving high message rates," in *13th IEEE/ACM International Symposium on Cluster Cloud and Grid Computing (CCGrid)*, May 2013, pp. 498–505.

[10] "EXTOLL homepage," http://www.extoll.de.

[11] H. Litz, H. Fröning, M. Nüssle, and U. Brüning, "VELO: A novel communication engine for ultra-low latency message transfers," in *37th International Conference on Parallel Processing (ICPP-08)*, Sept 2008, pp. 238–245.

[12] M. Nüssle, M. Scherer, and U. Brüning, "A resource optimized remote-memory-access architecture for low-latency communication," in *38th International Conference on Parallel Processing (ICPP-09)*, Sept 2009, pp. 220–227.

[13] H. Fröning and H. Litz, "Efficient hardware support for the partitioned global address space," in *10th Workshop on Communication Architecture for Clusters (CAC2010), co-located with 24th International Parallel and Distributed Processing Symposium (IPDPS 2010)*, April 2010, pp. 1–6.

[14] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, pp. 319–352, 1993.

[15] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, April 1999.

[16] M. Karol and M. Hluchyj, "Queuing in high-performance packet-switching," *IEEE Journal on Selected Areas*, vol. 1, pp. 1587–1597, 1998.

[17] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "The Tiny Tera: A packet switch core," *IEEE Micro*, vol. 17, pp. 27–33, 1997.

[18] Y. Tamir and G. Frazier, "High–performance multi–queue buffers for VLSI communications switches," *SIGARCH Comput. Archit. News*, vol. 16, no. 2, pp. 343–354, 1988.

[19] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[20] J. Duato, "A new theory of deadlock–free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, dec 1993.

[21] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for vlsi communication switches," *IEEE Trans. Comput.*, vol. 41, no. 6, pp. 725–737, Jun. 1992. [Online]. Available: http://dx.doi.org/10.1109/12.144624

[22] C. Carrion, R. Beivide, J. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proceedings of The Fourth International Conference on High-Performance Computing*, dec 1997, pp. 322–329.

[23] M. Atagoziyev, *Networks on Chip: Topology, Switching, Routing*. Saarbrücken, Germany: VDM Verlag, 2009.

[24] K. Bolding, "Non-uniformities introduced by virtual channel deadlock prevention," Tech. Rep., 1992.

[25] C. Izu, "Throughput fairness in k-ary n-cube networks," in *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ser. ACSC '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 137–145.