DE GRUYTER OPEN

Int. J. Appl. Math. Comput. Sci., 2015, Vol. 25, No. 4, 723–735
DOI: 10.1515/amcs-2015-0052

amcs

TORUS–CONNECTED CYCLES: A SIMPLE AND SCALABLE TOPOLOGY FOR INTERCONNECTION NETWORKS

ANTOINE BOSSARD [a,*], KEIICHI KANEKO [b]

[a]Graduate School of Science
Kanagawa University, Tsuchiya 2946, Hiratsuka, Kanagawa, 259-1293 Japan
e-mail: abossard@kanagawa-u.ac.jp

[b]Graduate School of Engineering
Tokyo University of Agriculture and Technology, 2-24-16 Nakacho, Koganei, Tokyo, 184-8588 Japan

Supercomputers are today made up of hundreds of thousands of nodes. The interconnection network is responsible for connecting all these nodes to each other. Different interconnection networks have been proposed; high performance topologies have been introduced as a replacement for the conventional topologies of recent decades. A high order, a low degree and a small diameter are the usual properties aimed for by such topologies. However, this is not sufficient to lead to actual hardware implementations. Network scalability and topology simplicity are two critical parameters, and they are two of the reasons why modern supercomputers are often based on torus interconnection networks (e.g., Fujitsu K, IBM Sequoia). In this paper we first describe a new topology, torus-connected cycles (TCCs), realizing a combination of a torus and a ring, thus retaining interesting properties of torus networks in addition to those of hierarchical interconnection networks (HINs). Then, we formally establish the diameter of a TCC, and deduce a point-to-point routing algorithm. Next, we propose routing algorithms solving the Hamiltonian cycle problem, and, in a two dimensional TCC, the Hamiltonian path one. Correctness and complexities are formally proved. The proposed algorithms are time-optimal.

Keywords: algorithm, routing, Hamiltonian, supercomputer, parallel.

1. Introduction

Today, supercomputers include hundreds of thousands of computing nodes. The one-million node barrier limit has even been broken recently by massively parallel systems such as China's Tianhe-2, as mentioned in the TOP500 list of June 2013 (TOP500, 2013). Aiming at the efficient operation of this huge amount of CPU nodes, many topologies have been described in the literature as being used as interconnection networks for massively parallel systems. However, the majority of the introduced network topologies have been facing hardware and/or software issues that have hampered the actual implementation and manufacture of these topologies.

Many network topologies that focus on the efficient connection of a large number of nodes have been proposed as detailed in the next section. However, as mentioned earlier, there remains a gap between these theoretical propositions and actual hardware implementations. Effectively, even if critical, the topological properties favoured by these networks, such as a high order, a low degree, and a small diameter, have proven insufficient to lead to actual hardware implementations. Hence, one can see that additional critical parameters play a determining role when it comes to hardware architecture decisions.

Network *scalability* and topology *simplicity* can be cited as two of these critical parameters. Also it is no coincidence that torus interconnects are very popular as interconnection networks of modern supercomputers—torus based networks indeed feature high scalability and simplicity. In the early days of supercomputing, hypercubes (Seitz, 1985) were popular due to their simplicity. However, due to the huge number of nodes involved in current massively parallel systems, this topology is no longer applicable. It is worth noting that this critical property (i.e., simplicity) has been largely relegated in modern interconnection

*Corresponding author

network proposals. In addition, modern supercomputers emphasise the need for expandability, for instance, to augment the computing power by plugging-in new computing nodes (see computer clusters). And again, scalability and simplicity are two reasons for the absence of hardware implementation for previously mentioned complex topologies.

It is thus reasonable to make a trade-off between the order/diameter of a network and its scalability/simplicity (i.e., ease of implementation). Hardware manufacturers are actually making similar decisions regarding the design of their parallel systems.

In this paper, we propose a new network topology, torus-connected cycles (TCCs), combining a torus for all its advantages, and an additional layer based on cycles that enables the expansion of the network order while retaining its low diameter; this is a hierarchical network. Then, we describe a point-to-point routing algorithm inside this network topology. Next, we present TCC routing algorithms that solve the Hamiltonian cycle problem, and, in a two dimensional TCC, the Hamiltonian path problem. Correctness and complexities of the proposed algorithms are formally proved. In addition, the proposed algorithms are time-optimal.

The rest of this paper is organised as follows. The state of the art is given in Section 2. The torus-connected cycles network topology is defined in Section 3, where the notation and definitions are also included. The network diameter of a TCC is established in Section 4. A point-to-point routing algorithm inside a TCC is subsequently deduced. An algorithm generating a Hamiltonian cycle in a TCC is proposed in Section 5. Then, in Section 6, an algorithm generating a Hamiltonian path in a two dimensional TCC is described. Finally, Section 7 concludes this paper.

2. State of the art

As stated in the introduction, due to their simplicity, hypercubes (Seitz, 1985) were popular with supercomputers of the previous century that included few nodes compared to modern machines such as China's Tianhe-2. Hypercubes still remain actively researched (Lai, 2012) though, for instance, as a seed network for more advanced topologies. However, now that the number of processing nodes has literally exploded, several network topologies targeted at massively parallel systems have been proposed. In general, such topologies focus on the connection of a large number of nodes while at the same time retaining a low degree and a small diameter. In particular, hierarchical interconnection networks (HINs) have proven very popular. Amongst those, dual-cubes (Li *et al.*, 2004) are actively researched: as examples, Zhou *et al.* (2012a) discussed dual-cube conditional fault diagnosability, and Shih *et al.* (2010) give a constructive proof of the existence of Hamiltonian cycles in dual-cubes.

Extending dual-cubes, Li *et al.* (2010) proposed metacubes, and routing algorithms such as that by Bossard *et al.* (2010) were set forward. Hierarchical cubic networks of Ghose and Desai (1995) are another example of HINs, with routing algorithms proposed by Bossard and Kaneko (2012a; 2013). Another famous HIN is the hierarchical hypercube of Malluhi and Bayoumi (1994) (also known as cube-connected cubes (Wu and Sun, 1994)), with routing algorithms proposed by Bossard *et al.* (2011; 2012b) and conditional fault diagnosability discussed by Zhou *et al.* (2012b). In addition, let us also cite cube-connected cycles of Preparata and Vuillemin (1981) since they are related to our research too. Now, as mentioned earlier, there remains a gap between these theoretical propositions and actual hardware implementations.

Hierarchical interconnection networks that are based on a torus are quite rare: one can merely mention the symmetric tori connected torus network (Al Faisal and Rahman, 2009), H3D-torus (Horiguchi and Ooki, 2000) and twisted torus (Camara *et al.*, 2010) as examples. This can be mainly explained by the difficulty in establishing the diameter of such networks, and thus shortest path routing algorithms. Nonetheless, interconnection networks based on a torus are very popular with modern parallel systems: amongst others, Fujitsu K and the Cray Titan are both built upon a torus-based network topology. Torus routing algorithms are indeed actively researched (Singh *et al.*, 2003; Xiang and Luo, 2012). In this paper, aiming at a reasonable network size and hardware applicability, we propose a torus-connected cycles topology with a much higher dimensional flexibility than in existing works, such as the H3D-torus, which focuses on two or three dimensions only.

3. Preliminaries

The main purpose of this section is to define a torus-connected cycles network topology. In addition, we present the notation used in this paper.

First, let us recall the topology definitions of a mesh network and of a torus network.

Definition 1. (*Mesh*) (*Duato* et al., *2003*) An n-dimensional mesh has k_i nodes on the i-th dimension, where $k_i \geq 2$ and $0 \leq i \leq n$, thus resulting in $k_0 \times k_1 \times \ldots \times k_{n-1}$ nodes in total. A node u is addressed with n coordinates $(u_0, u_1, \ldots, u_{n-1})$. Two nodes u, v are adjacent if and only if $u_i = v_i$ holds for all $i, 0 \leq i \leq n-1$, except one, j, where either $u_j = v_j + 1$ or $u_j = v_j - 1$.

Definition 2. ((k,n)-*torus*) (*Duato* et al., *2003*) A

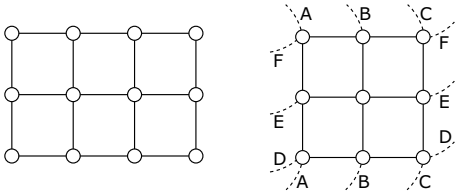


Fig. 1. 4×3 2-dimensional mesh (a), (3, 2)-torus (b).

k-ary n-dimensional torus, also called a (k, n)-torus, is an n-dimensional mesh with all k_i's equal to k and with wrap-around edges: two nodes u, v are adjacent if and only if $u_i = v_i$ holds for all i, $0 \leq i \leq n-1$, except one, j, where either $u_j = v_j + 1 \pmod k$ or $u_j = v_j - 1 \pmod k$ holds.

One can note that the degree of an (k, n)-torus is n if $k = 2$, and $2n$ otherwise. A 2-dimensional mesh of arities 3 and 4, and a (3, 2)-torus are illustrated in Fig. 1.

Definition 3. ($TCC(k, n)$) A k-ary n-dimensional torus-connected cycles network $TCC(k, n)$ is an undirected graph that has $2nk^n$ nodes. Each node $\boldsymbol{a}$ has a cluster ID $\boldsymbol{c}(\boldsymbol{a}) = (a_0, a_1, \ldots, a_{n-1})$ and a processor ID (also known as procID) $p(\boldsymbol{a}) = p_{\boldsymbol{a}}$, and the node consists of the pair $(\boldsymbol{c}(\boldsymbol{a}), p(\boldsymbol{a}))$, where $0 \leq a_i \leq k-1$ and $0 \leq p_a \leq 2n-1$. Each node $\boldsymbol{a}$ has three neighbour nodes $\boldsymbol{n}_1(\boldsymbol{a})$, $\boldsymbol{n}_2(\boldsymbol{a})$ and $\boldsymbol{n}_3(\boldsymbol{a})$ as follows:

$$\begin{aligned}
\boldsymbol{n}_1(\boldsymbol{a}) &= (\boldsymbol{c}(\boldsymbol{a}), (p_{\boldsymbol{a}} + (-1)^{p_{\boldsymbol{a}}}) \bmod 2n),\\
\boldsymbol{n}_2(\boldsymbol{a}) &= (\boldsymbol{c}(\boldsymbol{a}), (p_{\boldsymbol{a}} - (-1)^{p_{\boldsymbol{a}}}) \bmod 2n),\\
\boldsymbol{n}_3(\boldsymbol{a}) &= (a_0, a_1, \ldots, (a_{\lfloor p_{\boldsymbol{a}}/2 \rfloor} + (-1)^{p_{\boldsymbol{a}}})\\
&\qquad \bmod k, \ldots, a_{n-1}, p_{\boldsymbol{a}} + (-1)^{p_{\boldsymbol{a}}}).
\end{aligned}$$

If $n = 1$, each node $\boldsymbol{a}$ is of the degree two since $\boldsymbol{n}_1(\boldsymbol{a}) = \boldsymbol{n}_2(\boldsymbol{a})$. Otherwise, that is, $n \geq 2$, each node is of the degree three. For any node $\boldsymbol{a}$, the edges $(\boldsymbol{a}, \boldsymbol{n}_1(\boldsymbol{a}))$ and $(\boldsymbol{a}, \boldsymbol{n}_2(\boldsymbol{a}))$ are called *internal* edges whereas the edge $(\boldsymbol{a}, \boldsymbol{n}_3(\boldsymbol{a}))$ is called the *external* edge. So, each node $\boldsymbol{a}$ of a TCC has two internal neighbours: $\boldsymbol{n}_1(\boldsymbol{a})$ and $\boldsymbol{n}_2(\boldsymbol{a})$, and one external neighbour: $\boldsymbol{n}_3(\boldsymbol{a})$. If one deletes all external edges from a $TCC(k, n)$, the remaining graph consists of n^k disjoint cycles. Each cycle has $2n$ nodes. The cycle that includes a node $\boldsymbol{a}$ is denoted by $C(\boldsymbol{a})$, that is, the cluster of $\boldsymbol{a}$. In addition, if we contract each of the n^k cycles of a $TCC(k, n)$ into one single node, we obtained a k-ary n-dimensional torus (also known as (k, n)-torus).

Figure 2 shows an example of a 3-ary 2-dimensional torus-connected cycles network $TCC(3, 2)$. For example, the node (0, 3, 3) is connected to the nodes (0, 0, 0) and (0, 0, 2) by internal edges, and connected to the node (0, 2, 2) by an external edge.

In addition, we say that a node $\boldsymbol{a} = (c(\boldsymbol{a}), p_{\boldsymbol{a}})$ *covers* the dimension $\lfloor p_{\boldsymbol{a}}/2 \rfloor$. And so, the two nodes $\boldsymbol{a}$ and $\boldsymbol{n}_1(\boldsymbol{a})$ cover the same dimension $\lfloor p_{\boldsymbol{a}}/2 \rfloor$. For example, in a $TCC(3, 2)$, the nodes $(0, 0, 3)$ and $(0, 0, 2)$ cover the dimension 1, that is, of unit vector $(0, 1)$. For two nodes u and v covering the same dimension, we call u the counterpart (or twin) of v, and vice-versa.

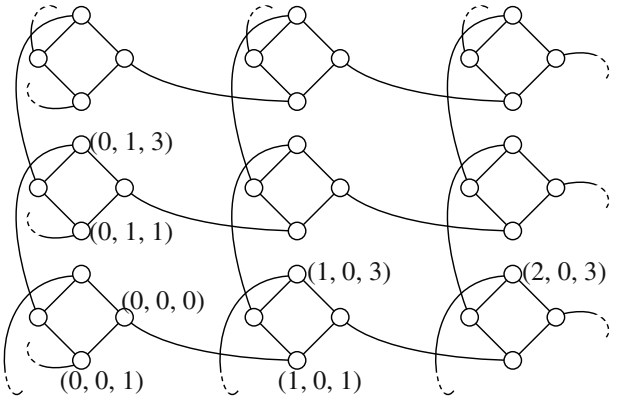


Fig. 2. Example of a TCC(3, 2): 4-cycles are connected according to a (k, n)-torus.

Lemma 1. *A $TCC(k, n)$ is a bipartite graph.*

Proof. Assume that the processor IDs of nodes are numbered $0, 1, 2 \ldots$ similarly in each cluster. Then it is easy to see that each node of an even (resp. odd) processor ID is connected exclusively to nodes with odd (resp. even) processor IDs since clusters contain an even number of nodes: $2n$. ■

Lastly, let us recall several general definitions and the notation for graphs. A path in a graph is an alternate sequence of distinct nodes and edges $\boldsymbol{u}_0, (\boldsymbol{u}_0, \boldsymbol{u}_1), \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{k-1}, (\boldsymbol{u}_{k-1}, \boldsymbol{u}_k), \boldsymbol{u}_k$, which can be written simply as $\boldsymbol{u}_0 \to \boldsymbol{u}_1 \to \ldots \to \boldsymbol{u}_k$, and abbreviated to $\boldsymbol{u}_0 \rightsquigarrow \boldsymbol{u}_k$. The length of a path corresponds to the number of edges included in that path; in the previous example, the path is of length k. In a graph, we say that two nodes are diagonally opposed if and only if the length of a shortest path connecting them is equal to the graph diameter, that is, the maximum length of a shortest path between any two nodes.

4. Diameter and simple routing

In this section, we establish the diameter of a $TCC(k, n)$. The constructive proof described can subsequently be used as a point-to-point routing algorithm. First, it is important to note that routing inside a TCC is closely related to routing inside a cycle. Effectively, each of the n dimensions of a TCC is iterated by traversing the corresponding nodes inside cycles (i.e., clusters) of a TCC. Thus, we can reduce the point-to-point routing problem in a TCC to the traversal of one unique $2n$-cycle:

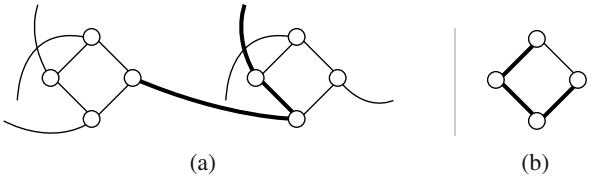


Fig. 3. TCC routing (a) can be reduced to the cycle traversal (b).

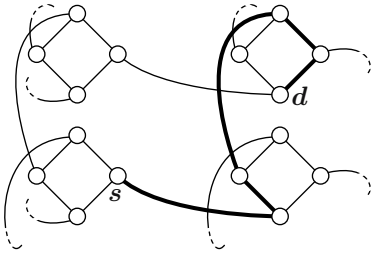


Fig. 4. Constructing a shortest path in a $TCC(2,2)$.

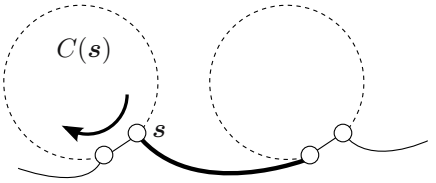


Fig. 5. Cycle traversal direction induced by the selection of the external edge incident with $\boldsymbol{s}$.

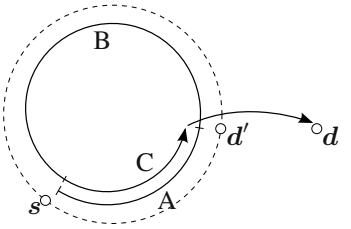


Fig. 6. Cycle traversal: $k/2-1$ external edges consumed on part A ($\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$), $k/2$ on part B ($\boldsymbol{d}' \rightsquigarrow \boldsymbol{s}$) and 1 on part C ($\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$).

the dimension traversed by one external edge corresponds to the internal edge on the cycle between the two nodes that cover that same dimension; see Fig. 3.

4.1. Formally established diameter. Three cases are distinguished: $k=2$, $k \geq 3$ with k even, and $k \geq 3$ with k odd.

Lemma 2. *A $TCC(2,n)$ (i.e., $k=2$) has diameter 5 in the case $n=2$, and at most $3n-2$ in the case $n \geq 3$.*

Proof. We give a constructive proof. Two cases are distinguished: $n=2$ and $n \geq 3$. Let us consider two nodes $\boldsymbol{s}$ and $\boldsymbol{d}$ that are diagonally opposed.

If $n=2$, select the external edge $\boldsymbol{s} \to \boldsymbol{s}'$ incident with $\boldsymbol{s}$; then select the internal edge $\boldsymbol{s}' \to \boldsymbol{u}$ such that the external edge of $\boldsymbol{u}$ is connected to $C(\boldsymbol{d})$, say $\boldsymbol{u} \to \boldsymbol{v} \in C(\boldsymbol{d})$. Lastly, we traverse half of $C(\boldsymbol{d})$ to reach $\boldsymbol{d}$, thus requiring 2 internal edges. In total, the selected path $\boldsymbol{s} \to \boldsymbol{s}' \to \boldsymbol{u} \to \boldsymbol{v} \rightsquigarrow \boldsymbol{d}$ is of length 5; see Fig. 4.

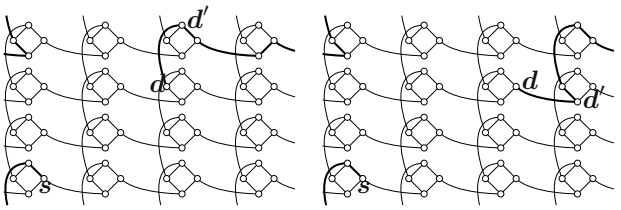
If $n \geq 3$, select the external edge incident with $\boldsymbol{s}$; this induces a cycle traversal direction (see Fig. 5). Continue traversing the cycle in this direction so as to consume 1 external edge for each of the $n-1$ remaining dimensions, finally reaching $C(\boldsymbol{d})$; this takes $1+2(n-1)$ edges. Lastly, $\delta \leq n$ internal edges in $C(\boldsymbol{d})$ are required to reach $\boldsymbol{d}$. If $\delta = n$, then we discard the selected path and start again by traversing the cycle in the opposite direction (i.e., we first select the internal edge incident with $\boldsymbol{s}$ and its counterpart). Because this time the $n-1$ dimensions are consumed in reverse order compared with the previous selection, we arrive in $C(\boldsymbol{d})$ at the diagonally opposed position as previously, and thus $\delta \leq n-2$ (actually $\delta = 0$). So, in total, the selected path is of length at most $1+2(n-1)+(n-1) = 3n-2$. Note that if $\delta = n$ in the first place, $\boldsymbol{s}$ and $\boldsymbol{d}$ are not diagonally opposed since they can be connected with the second path in $2n$ edges. ∎

Lemma 3. *A $TCC(k,n)$ with $k \geq 3$ and k even has diameter nk.*

Proof. We give a constructive proof. Let us first consider a (k,n)-torus. If k is even, two diagonally opposed nodes are separated by the same minimum number of edges, regardless of the direction in which you traverse the torus to join these two nodes. Thus, in a $TCC(k,n)$, one can freely choose the rotation direction inside the cycle without impacting the number of external edges needed to reach a diagonally opposed cycle. Below we give a shortest path routing algorithm that connects two diagonally opposed nodes in a TCC; see Fig. 6 and examples in Fig. 7.

Consider two special positions on the cycle: the position of the source node $\boldsymbol{s}$ and that of the external neighbour $\boldsymbol{d}'$ of $\boldsymbol{d}$. Traverse the cycle in the direction where the position of $\boldsymbol{d}'$ is reached before the counterpart position for the same dimension. By traversing the cycle according to this direction, consume $k/2-1$ external edges per dimension for the dimensions on the $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ part of the cycle, $\boldsymbol{d}'$ dimension included. Then, continue the cycle traversal by consuming $k/2$ external edges per dimension for the dimensions on the $\boldsymbol{d}' \rightsquigarrow \boldsymbol{s}$ part of the cycle, this time with $\boldsymbol{d}'$ dimension not included. Finally, achieve the cycle traversal by consuming 1 external edge per dimension for the dimensions on the $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ part of the cycle. This way, we are guaranteed that no internal edge will ever be selected inside $C(\boldsymbol{d})$.

Consuming one external edge on one dimension requires 1 internal edge on the cycle and 1 external



(a) (b)

Fig. 7. Two cases of diagonally opposed nodes $\boldsymbol{s}, \boldsymbol{d}$ connected with shortest paths (length 8) (a), (b).

edge. So, on the $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ part of the cycle, assuming δ dimensions are consumed on this part, we have $\delta \times 2(k/2-1)$ edges required. We recall that dimensions on this part of the cycle are consumed up to $k/2-1$ external edges. Then, on the $\boldsymbol{d}' \rightsquigarrow \boldsymbol{s}$ part of the cycle, $n-\delta$ dimensions are fully consumed (i.e., up to $k/2$ external edges); it takes $(n-\delta) \times 2(k/2)$ edges. Finally, on the part $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ of the cycle, δ dimensions are consumed with only one external edge for each dimension; it takes $\delta \times 2(1)$ edges. In total we have found a path between two diagonally opposed nodes of length

$$\delta \times 2(k/2-1) + (n-\delta) \times 2(k/2) + \delta \times 2(1) = nk,$$

which is obviously a shortest path. ■

Lemma 4. *A $TCC(k,n)$ with $k \geq 3$ and k odd has diameter at most $nk+n$.*

Proof. We give a constructive proof. Again, let us first consider a (k,n)-torus. If k is odd, depending on the direction used to traverse the torus to join two diagonally opposed nodes, the minimum number of edges required varies. Considering one single dimension, depending on the direction used to traverse this dimension, either $\lfloor k/2 \rfloor$ or $\lceil k/2 \rceil$ edges are required to reach a diagonal node. Thus, in a $TCC(k,n)$, the direction used to traverse the cycle will impact on the number of external edges needed to attain a diagonally opposed cluster. So, to find a path between two diagonally opposed nodes $\boldsymbol{s}$ and $\boldsymbol{d}$, we apply the same idea as in the case k even. However, we take care to stay at a distance of 1 external edge to $C(\boldsymbol{d})$ for dimensions of $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$, $\boldsymbol{d}'$ included, and this may induce selecting successive internal edges to skip dimensions. Assuming that each of all dimensions is consumed in the direction requiring $(k+1)/2$ external edges, the maximum length of a generated path is as follows.

On the $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ part of the cycle, assuming δ dimensions are consumed on this part, we have $\delta \times 2((k-1)/2)$ edges required. We recall that dimensions on this part of the cycle are consumed up to $(k-1)/2$ external edges. Then, on the $\boldsymbol{d}' \rightsquigarrow \boldsymbol{s}$ part of the cycle, $n-\delta$ dimensions are fully consumed (i.e., up to $(k+1)/2$ external edges), taking $(n-\delta) \times 2((k+1)/2)$ edges. Finally, on the part $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'$ of the cycle, δ dimensions are consumed with only one external edge for each dimension, taking $\delta \times 2(1)$ edges. In total, we have found a path between two diagonally opposed nodes of length:

$$\delta \times 2((k-1)/2) + (n-\delta) \times 2((k+1)/2) + \delta \times 2(1) = nk+n.$$

■

Table 1. Empirically calculated diameter of a $TCC(k,n)$.

n \ k	2	3	4	5	6	7	8
2	5	6	8	10	12	14	16
3	7	9	12	15	18	21	24
4	10	13	16	20	24	28	32
5	13	16	20	25	30		
6	16	20	24	30	36		
7	19	23	28				
8	22	27					
9	25						
10	28						

An extra cost of n edges compared with the case where k is even is induced; however, this has very low actual impact. Effectively, the dimension of a torus (n) is always very small compared to its arity (k); for example, the Cray Titan uses a three dimensional torus to connect hundreds of thousands of nodes, i.e., $n \ll k$.

Thus, for any two TCC nodes $\boldsymbol{u}, \boldsymbol{v}$, that is, two nodes that are not necessarily diagonally opposed, we can apply the same algorithm to find a path $\boldsymbol{u} \rightsquigarrow \boldsymbol{v}$ whose length is at most the diameter as established previously. We summarise this discussion in Theorem 1 below.

Theorem 1. *A $TCC(k,n)$ has diameter 5 for $k=2, n=2$; at most $3n-2$ for $k=2, n \geq 3$; nk in the case k is even, and at most $nk+n$ in the case k is odd.*

Proof. This can be directly deduced from Lemmas 2–4. ■

4.2. Experimentation and comparison with related networks. We have empirically calculated the diameter of a $TCC(k,n)$ by finding all the possible shortest paths between all the possible pairs of nodes. Then, considering all the shortest paths established, we have retained the length of the longest one, that is, the diameter of the TCC for this instance of k, n. Results are given in Table 1.

We observe that the case $k=2$ is indeed special as detailed in Lemma 2. As for the cases $k \geq 3$, we see that the empirical data are matching our theoretical estimations of nk in the case k is even, and at most $nk+n$ otherwise.

Finally, we summarise in Table 2 the TCC topological properties and compare them with those of

Table 2. Comparing TCCs, CCCs and (k, n)-tori.

	$CCC(n)$	$TCC(k, n)$	(k, n)-torus
order	$n \times 2^n$	$2n \times k^n$	k^n
degree	3	3	$2n$
diameter	$2n + \lfloor n/2 \rfloor - 2$	nk	$n\lfloor k/2 \rfloor$
bisection	2^{n-1}	$2 \times k^{n-1}$	$2 \times k^{n-1}$

related networks, namely, (k, n)-tori (Duato *et al.*, 2003) and cube-connected cycles (Preparata and Vuillemin, 1981). A cube-connected cycle $CCC(d, k)$ is a d-cube connecting 2^d k-cycles. A $CCC(d, k)$ with $d \neq k$ is not considered here since it is neither symmetric nor regular. So, we consider only the case $d = k$, and simply denote $CCC(n)$. Additionally, for the sake of clarity, we only consider the case where k is even for a $TCC(k, n)$.

The bisection width of a graph G is the cardinality of a minimum set of edges H such that $G \setminus H$ is made of two disconnected subgraphs of the same size. The bisection width is an important metric regarding fault-tolerance. We observe that the TCC bisection width is the same as that of a (k, n)-torus: k edges on each of the $n-1$ dimensions are cut in two places so that the initial graph is separated into two similar entities.

5. Hamiltonian cyclicity

In this section we show the existence of a Hamiltonian cycle in a TCC by giving a constructive proof. This fundamental routing problem has many important applications. The main idea of this algorithm is to follow a divide-and-conquer approach: the Hamiltonian cycle problem in a $TCC(k, n)$ is reduced to the same problem in two $TCC(k, n-1)$ networks. We proceed recursively as follows (see Fig. 8).

First, the base case is given in Lemma 5 below.

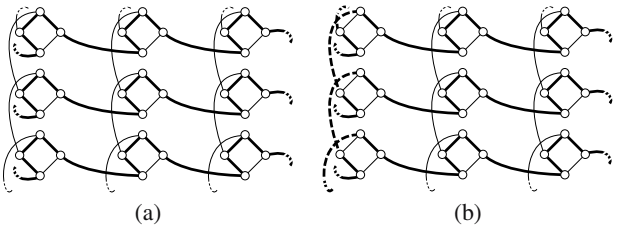
Lemma 5. *In a $TCC(k, 1)$ (i.e., $n = 1$), we can find a Hamiltonian cycle in $O(k)$ optimal time.*

Proof. Ignoring the extra nodes inside clusters, if any, a $TCC(k, 1)$ is isomorphic to a ring, and thus it is trivial to include extra nodes inside clusters to obtain a Hamiltonian cycle. A $TCC(k, 1)$ has $2k$ nodes, and thus the time complexity to construct a Hamiltonian cycle is $O(k)$, which is obviously optimal. ■

Then, we have the general case reducing towards the base case of Lemma 5.

Lemma 6. *In a $TCC(k, n)$ with $n \geq 2$, we can find a Hamiltonian cycle in $O(kn)$ optimal time.*

Proof. We give a constructive proof. Select a dimension i ($0 \leq i \leq n-1$) and divide the $TCC(k, n)$ into k sub-networks $TCC(k, n-1)$ along the dimension i. When reducing from a $TCC(k, n)$ into sub-networks



(a) (b)

Fig. 8. First: constructing distinct Hamiltonian cycles inside k $TCC(k, n-1)$ sub-networks (a). Second: connecting the k distinct Hamiltonian cycles together (b).

$TCC(k, n-1)$, clusters remain $2n$-cycles (as in a $TCC(k, n)$); we simply assume that such a $TCC(k, n-1)$ has extra nodes inside clusters, but these nodes have no impact since they are incident with only internal edges. Apply this algorithm recursively to obtain a Hamiltonian cycle inside each of the k sub-networks $TCC(k, n-1)$.

Inside one of the k sub-networks $TCC(k, n-1)$, we select one internal edge that is incident with the two nodes covering the dimension i. There are kn such edges, one per cluster of the $TCC(k, n-1)$. This selected edge, as well as those corresponding to it inside the other $TCC(k, n-1)$ sub-networks, are discarded from the Hamiltonian cycles previously obtained recursively. In total, k internal edges are thus discarded. The external edges incident with the nodes of the discarded edges are selected in order to connect the k distinct Hamiltonian cycles together, and thus obtain a Hamiltonian cycle in the $TCC(k, n)$.

Regarding the time complexity, let N be the dimension of the original network (i.e., $TCC(k, N)$); the algorithm starts with $n = N$. In the case $n = 1$, the Hamiltonian cycle is trivial and returned in $O(kN)$ so as to include all extra nodes inside clusters (in total, $2N$ nodes per cluster). The selection of the dimension i is performed in constant time. Then, this is a recursive call: it is processed n times. The remaining operations are $O(k)$ time. So, in total, this algorithm takes $O(kN)$ optimal time. ■

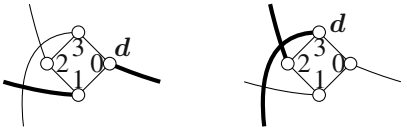
We summarise this discussion in Theorem 2 below.

Theorem 2. *In a $TCC(k, n)$, we can construct a Hamiltonian cycle in $O(kn)$ optimal time.*

Proof. It can be directly deduced from Lemmas 5 and 6. ■

6. Hamiltonian laceability of a $TCC(k, 2)$ and a Hamiltonian path algorithm

In this section, we show the Hamiltonian laceability of a $TCC(k, 2)$ by giving a constructive proof. A k-ary 2-dimensional TCC offers the widest range of applications due to its simple, yet powerful (regularity, scalability,



(a) (b)

Fig. 9. Dimension covered by $\boldsymbol{d}$: either horizontal (a) or vertical (b).

symmetry, etc.), structure on both the high-level (torus level) and the low-level (clusters).

From Lemma 1, a $TCC(k, n)$ is bipartite. So clearly, there does not exist a Hamiltonian path for any pair of nodes. Precisely, any two nodes can be connected by a Hamiltonian path if and only if they are located in distinct partitions of the bipartite graph. We say that a graph satisfying this property is *Hamiltonian laceable*. So, in a $TCC(k, n)$, any two nodes can be connected by a Hamiltonian path if and only if they have processor IDs of distinct parities. Hence, we show here that a $TCC(k, 2)$ is Hamiltonian laceable, and we give an algorithm which, given any two nodes whose processor IDs are of distinct parities, finds a Hamiltonian path (Hamiltonian lace) between them.

In a $TCC(k, 2)$, for any two nodes $\boldsymbol{s}$ and $\boldsymbol{d}$ of distinct parities, we describe in this section an algorithm establishing a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$. Since a TCC is symmetric, we can assume without loss of generality that $\boldsymbol{s} = (0, 0, 0)$. The main idea of this algorithm is to select a traversing order depending on the dimension covered by $\boldsymbol{d}$: either horizontal or vertical, see Fig. 9. Assuming that a horizontal traversal has been selected, we then traverse successively each of the k rows of k clusters, finally reaching $C(\boldsymbol{d})$, the cluster of $\boldsymbol{d}$ via $C(\boldsymbol{u})$, the cluster containing the external neighbour $\boldsymbol{u}$ of $\boldsymbol{d}'$ the twin node of $\boldsymbol{d}$.

We recall that, since $n = 2$, each cluster is a 4-cycle. For the sake of clarity, we assume without loss of generality that processor IDs inside clusters are numbered as follows: 0 – East, 1 – South, 2 – West and 3– North, and that processor IDs 0 and 1 (resp. 2 and 3) are for the horizontal (resp. vertical) dimension, that is, as shown in Fig. 9.

We distinguish two main cases: $\boldsymbol{d}$ on the horizontal dimension, that is, $p(\boldsymbol{d}) = 1$, since we assumed that $p(\boldsymbol{s}) = 0$, and we recall that $\boldsymbol{s}$ and $\boldsymbol{d}$ must have processor IDs of distinct parities, and $\boldsymbol{d}$ on the vertical dimension, that is, $p(\boldsymbol{d}) = 3$, for the same reasons.

Lemma 7. *In a $TCC(k, 2)$, for $\boldsymbol{s} = (0, 0, 0)$ and a node $\boldsymbol{d} \notin C(\boldsymbol{s})$ with procIDs of distinct parities, and with $\boldsymbol{d}$ on the horizontal dimension (i.e., $p(\boldsymbol{d}) = 1$), we can find a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$.*

Proof. We give a constructive proof. As $\boldsymbol{d}$ is on the horizontal dimension, we traverse the network vertically, one column after another. The special cases where $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension are solved in Appendices A (the dimension of $\boldsymbol{d}$ is shared) and B (the dimension of $\boldsymbol{d}$ is not shared).

Step 1. We initialise the path for a vertical traversal of the network; see Fig. 10(a).

Step 2. Extend the path obtained in Step 1 by traversing each of the $k - 1$ remaining columns of clusters. In practice, this is done by deselecting inside each cluster of the row of $C(\boldsymbol{s})$ except $C(\boldsymbol{s})$ the edge incident with the two nodes of the vertical dimension (i.e., nodes of processor IDs 2 and 3), and instead traverse the corresponding cluster column; see Fig. 11(a). If $\boldsymbol{h} = \boldsymbol{d}$, a Hamiltonian lace is found.

Step 3. First, in $C(\boldsymbol{d})$, replace the original full traversal $2, 1, 0, 3$ by $2, 3$ so that the path does not go through $\boldsymbol{d}$ yet. Define $C(\boldsymbol{u})$ the cluster containing the external neighbour $\boldsymbol{u}$, of $\boldsymbol{d}'$ the twin node of $\boldsymbol{d}$. Formally, $\boldsymbol{d}' = n_1(\boldsymbol{d})$, and $\boldsymbol{u} = n_3(\boldsymbol{d}')$. Replace the original full traversal $2, 1, 0, 3$ by $2, 3$ so that the path does not go through $\boldsymbol{u}$ yet.

Find a cluster C_1 that satisfies the following conditions:

- C_1 must be in the same column as $C(\boldsymbol{h})$ with $\boldsymbol{h}$, the head of the path that is being constructed, and strictly before $C(\boldsymbol{h})$ (i.e., $C(\boldsymbol{h}) \neq C_1$) according to the traversal direction (i.e., here, above $C(\boldsymbol{h})$).
- C_1 must not be in the same row as $C(\boldsymbol{s})$.
- C_1 can be on the same horizontal dimension as $C(\boldsymbol{d})$ (i.e., same row), or below $C(\boldsymbol{d})$, but not above $C(\boldsymbol{d})$.

We say that C_1 is the cluster for the first elbow. In a few special cases, there is no such cluster C_1; these cases are treated separately in Appendices A and B.

Step 4. If C_1 is in the same row as $C(\boldsymbol{u})$, that is, C_1 is at the crossing of $C(\boldsymbol{u})$ and $C(\boldsymbol{h})$, go directly to Step 5. Otherwise, select the cluster C_2 that is in the same column as $C(\boldsymbol{u})$ and in the same row as C_1. We say that C_2 is the cluster for the second elbow.

Step 5. We modify the traversing of $C(\boldsymbol{h})$ so that $C(\boldsymbol{h})$ is now fully traversed at once: $0, 1, 2, 3$ $(= \boldsymbol{h})$ instead of $0, 3 \rightsquigarrow 2, 1$ $(= \boldsymbol{h})$. Then, from this new $\boldsymbol{h}$, we follow the previously established path until reaching C_1. Once C_1 is attained (we arrive at procID 2), we modify the traversing of C_1 so that the path exits C_1 via 1 instead of 3: C_1 is now traversed in the order $2, 3, 0, 1$ instead of $2, 1, 0, 3$.

Step 6. Now, we will traverse the network horizontally, thus having the path being constructed enter several clusters twice, mixing the original vertical traversal and a horizontal traversal to eventually reach $C(\boldsymbol{u})$. In

practice, we exit C_1 via procID 1 and arrive in the next cluster, say C_1', on procID 0. The path obtained previously fully traversed C_1' in the order $2,1,0,3$; this traversal of C_1' is now split into two parts: the original path for a vertical traversal now shrinks to $2,3$ instead of $2,1,0,3$, and the horizontal traversal arriving from C_1 is set to $0,1$.

Apply the same method of having the path being constructed enter each cluster twice in the row of C_1 until reaching C_2, or directly $C(\boldsymbol{u})$ if C_2 has not been defined. If $C(\boldsymbol{u})$ is reached, go directly to Step 8.

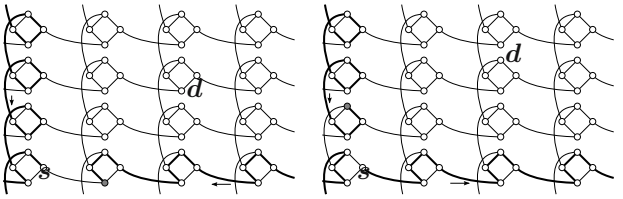
Step 7. We have arrived in C_2 on procID 0. In the path previously established, C_2 is fully traversed in the order $2,1,0,3$; we replace this traversal by $0,1,2,3$. Then, we apply the same method as in Step 6 until reaching $C(\boldsymbol{u})$, that is, having the path being constructed enter clusters twice in the column of C_2 between C_2 and $C(\boldsymbol{u})$ in the original vertical traversal order (here downwards).

Step 8. If $C(\boldsymbol{u})$ is reached in Step 6, then we arrive in $C(\boldsymbol{u})$ on procID 0. We traverse $C(\boldsymbol{u})$ for the second time according to $0,1\ (=\boldsymbol{u})$, and we finally reach $\boldsymbol{d}$ with the sub-path $\boldsymbol{u} \to \boldsymbol{d}' \to \boldsymbol{d}$. See Fig. 13(a).

Now, if $C(\boldsymbol{u})$ is reached in Step 7, then we arrive in $C(\boldsymbol{u})$ on procID 2. Instead of the original $2,1,0,3$ traversal of $C(\boldsymbol{u})$, we now follow the order $2,3,0,1\ (=\boldsymbol{u})$. Finally, we reach $\boldsymbol{d}$ with the sub-path $\boldsymbol{u} \to \boldsymbol{d}' \to \boldsymbol{d}$; see Fig. 12(a).

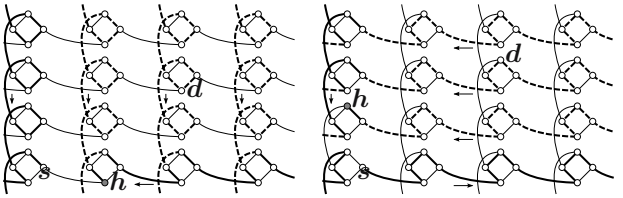
Step 9. Now, consider the column of C_1, a column which by definition also includes $C(\boldsymbol{h})$. In the case C_1 is not adjacent to $C(\boldsymbol{h})$ in the original vertical traversal order (here downwards), that is, there exists one or more clusters between C_1 and $C(\boldsymbol{h})$ in the original vertical traversal order, these clusters have been completely skipped from the previously established $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ path. The same remarks hold if C_2 is not adjacent to $C(\boldsymbol{u})$: clusters between $C(\boldsymbol{u})$ and C_2 according to the original vertical traversal order will be skipped. So, we shall now extend the established $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ path so that it includes the traversal of the clusters between C_1 and $C(\boldsymbol{h})$. For each such skipped cluster, we extend the path by traversing horizontally the row of this cluster, applying the same method as in Steps 6 and 7: clusters in this row will now be entered twice by the path, the first time for the original vertical traversal, and the second time for the horizontal traversal for path extension.

We detail the path extension in the case C_1 is not adjacent to $C(\boldsymbol{h})$. The same process can be applied to skipped clusters between $C(\boldsymbol{u})$ and C_2 (see Fig. A3(a)). Consider the row of each such skipped cluster $\tilde{C}$. In this row, consider the cluster $\tilde{C}'$ that is in the same column as $C(\boldsymbol{s})$. The path established previously traversed $\tilde{C}'$ in the order $2,1,0,3$. Now, instead of selecting the edge $1,0$ in $\tilde{C}'$, we traverse the row of $\tilde{C}'$ (horizontally, obviously) as follows. First, we exit $\tilde{C}'$ via procID 1. If the next cluster is $\tilde{C}$, then it is fully traversed according to $0,3,2,1$, and subsequently exited. If the next cluster is not $\tilde{C}$, then it will be entered by the path twice: previously fully traversed according to $2,1,0,3$, the original vertical traversal now shrinks to $2,3$, and the horizontal traversal currently being established selects $0,1$ and goes to the next cluster. Repeat this operation until returning to $\tilde{C}'$. In such a row, exactly one cluster ($\tilde{C}$) will be fully traversed by this horizontal traversal for path extension, and all other clusters in the row will be entered twice by the path being constructed. See Fig. 13(a).



(a) (b)

Fig. 10. Step 1: path initialisation for vertical (a) and horizontal (b) traversals. Newly selected edges in bold; the greyed node is the head of the (still in construction) path.



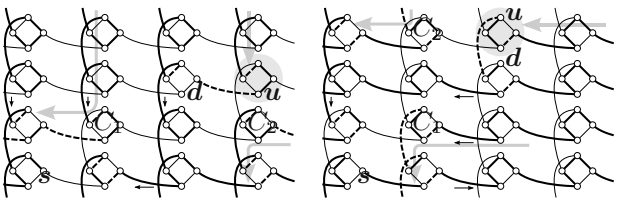
(a) (b)

Fig. 11. Step 2: path extension for vertical (a) and horizontal (b) traversals. Newly selected edges in dashes.

■

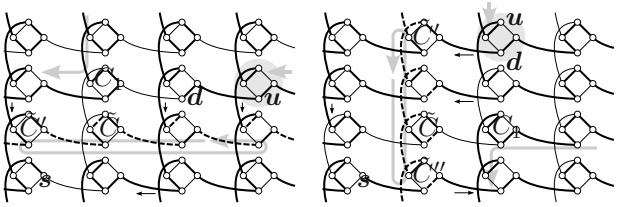
Lemma 8. *In a $TCC(k,2)$, for $\boldsymbol{s}=(0,0,0)$ and a node $\boldsymbol{d} \notin C(\boldsymbol{s})$ with procIDs of distinct parities, and with $\boldsymbol{d}$ on the vertical dimension (i.e., $p(\boldsymbol{d})=3$), we can find a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$.*

Proof. We give a constructive proof. As $\boldsymbol{d}$ is on the



(a) (b)

Fig. 12. Step 8: going through C_1 and C_2 before reaching $C(\boldsymbol{u})$ and eventually $\boldsymbol{d}$ in the case of vertical (a) and horizontal (b) traversals. Newly selected edges in dashes.



(a) (b)

Fig. 13. Step 9: extending the path to include skipped clusters $\tilde{C}$ (if any) in the case of vertical (a) and horizontal (b) traversals. Path extension emphasised by dashed edges.

vertical dimension, we traverse the network horizontally, one row after another. The algorithm is rather symmetrical to that of Lemma 7. Similarly, the special cases where $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension are solved in Appendices C (the dimension of $\boldsymbol{d}$ is shared) and D (the dimension of $\boldsymbol{d}$ is not shared).

Step 1. We initialise the path for a horizontal traversal of the network; see Fig. 10(b).

Step 2. Extend the path obtained in Step 1 by traversing each of the $k-1$ remaining rows of clusters. In practice, this is done by deselecting inside each cluster of the column of $C(\boldsymbol{s})$ except $C(\boldsymbol{s})$ the edge incident with the two nodes of the horizontal dimension (i.e., nodes of processor IDs 0 and 1), and traverse the corresponding cluster row instead; see Fig. 11(b). If $\boldsymbol{h} = \boldsymbol{d}$, a Hamiltonian lace is found.

Step 3. First, in $C(\boldsymbol{d})$, replace the original full traversal $0, 3, 2, 1$ by $0, 1$ so that the path does not go through $\boldsymbol{d}$ yet. Define $C(\boldsymbol{u})$ the cluster containing the external neighbour $\boldsymbol{u}$ of $\boldsymbol{d}'$, the twin node of $\boldsymbol{d}$. Formally, $\boldsymbol{d}' = n_1(\boldsymbol{d})$ and $\boldsymbol{u} = n_3(\boldsymbol{d}')$. Replace the original full traversal $0, 3, 2, 1$ by $0, 1$ so that the path does not go through $\boldsymbol{u}$ yet.

Find a cluster C_1 that satisfies the following conditions:

- C_1 must be in the same row as $C(\boldsymbol{h})$ with $\boldsymbol{h}$, the head of the path that is being constructed, and strictly before $C(\boldsymbol{h})$ (i.e., $C(\boldsymbol{h}) \neq C_1$) according to the traversal direction (i.e., here, on the right of $C(\boldsymbol{h})$).
- C_1 must not be in the same column as $C(\boldsymbol{s})$.
- C_1 can be on the same vertical dimension as $C(\boldsymbol{d})$ (i.e., same column), or on the left of $C(\boldsymbol{d})$, but not on the right of $C(\boldsymbol{d})$.

We say that C_1 is the cluster for the first elbow. In a few special cases, there is no such cluster C_1; these cases are treated separately in Appendices C and D.

Step 4. If C_1 is in the same column as $C(\boldsymbol{u})$, that is, C_1 is at the crossing of $C(\boldsymbol{u})$ and $C(\boldsymbol{h})$, go directly to Step 5. Otherwise, select the cluster C_2 that is in the same row as $C(\boldsymbol{u})$ and in the same column as C_1. We say that C_2 is the cluster for the second elbow.

Step 5. We modify the traversing of $C(\boldsymbol{h})$ so that $C(\boldsymbol{h})$ is now fully traversed at once: $2, 3, 0, 1$ $(= \boldsymbol{h})$ instead of $2, 1 \rightsquigarrow 0, 3$ $(= \boldsymbol{h})$. Then, from this new $\boldsymbol{h}$, we follow the previously established path until reaching C_1. Once C_1 is attained (we arrive in procID 0), we modify the traversing of C_1 so that the path exits C_1 via 3 instead of 1: C_1 is now traversed in the order $0, 1, 2, 3$ instead of $0, 3, 2, 1$.

Step 6. Now, we will traverse the network vertically, thus having the path being constructed enter twice several clusters, mixing the original horizontal traversal and a vertical traversal to eventually reach $C(\boldsymbol{u})$. In practice, we exit C_1 via procID 3 and arrive in the next cluster, say C_1', on procID 2. The path obtained previously fully traversed C_1' in the order $1, 2, 3, 0$; this traversal of C_1' is now split into two parts: the original path for horizontal traversal now shrinks to $1, 0$ instead of $1, 2, 3, 0$, and the vertical traversal arriving from C_1 is set to $2, 3$.

Apply the same method of having the path being constructed enter each cluster twice in the column of C_1 until reaching C_2, or directly $C(\boldsymbol{u})$ if C_2 has not been defined. If $C(\boldsymbol{u})$ is reached, go directly to Step 8.

Step 7. We have arrived in C_2 on procID 2. In the path previously established, C_2 is fully traversed in the order $0, 3, 2, 1$; we replace this traversal by $2, 3, 0, 1$. Then, we apply the same method as in Step 6 until reaching $C(\boldsymbol{u})$, that is, having the path being constructed enter clusters twice in the row of C_2 between C_2 and $C(\boldsymbol{u})$ in the original horizontal traversal order (here leftwards).

Step 8. If $C(\boldsymbol{u})$ is reached in Step 6, then we arrive in $C(\boldsymbol{u})$ on procID 2. We traverse $C(\boldsymbol{u})$ for the second time according to $2, 3$ $(= \boldsymbol{u})$ and we finally reach $\boldsymbol{d}$ with the sub-path $\boldsymbol{u} \rightarrow \boldsymbol{d}' \rightarrow \boldsymbol{d}$; see Fig. 13(b).

Now, if $C(\boldsymbol{u})$ is reached in Step 7, then we arrive in $C(\boldsymbol{u})$ on procID 2. Instead of the original $2, 1, 0, 3$ traversal of $C(\boldsymbol{u})$, we now follow the order $2, 3, 0, 1$ $(= \boldsymbol{u})$. Finally, we reach $\boldsymbol{d}$ with the sub-path $\boldsymbol{u} \rightarrow \boldsymbol{d}' \rightarrow \boldsymbol{d}$; see Fig. 12(b).

Step 9. Now, consider the row of C_1, a row which by definition also includes $C(\boldsymbol{h})$. In the case C_1 is not adjacent to $C(\boldsymbol{h})$ in the original horizontal traversal order (here leftwards), that is, there exists one or more clusters between C_1 and $C(\boldsymbol{h})$ in the original horizontal traversal order, these clusters have been completely skipped from the previously established $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ path. The same remarks hold if C_2 is not adjacent to $C(\boldsymbol{u})$: clusters between $C(\boldsymbol{u})$ and C_2 according to the original horizontal traversal order will be skipped. So, we shall now extend the established $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ path so that it includes the traversal of the clusters between C_1 and

$C(\boldsymbol{h})$. For each such skipped cluster, we extend the path by traversing vertically the column of this cluster, applying the same method as in Steps 6 and 7: clusters in this column will now be entered twice by the path, the first time for the original horizontal traversal, and the second time for the vertical traversal for path extension.

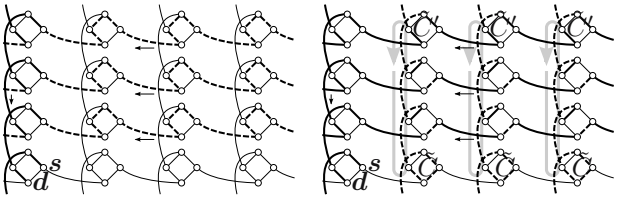
We detail the path extension in the case C_1 is not adjacent to $C(\boldsymbol{h})$. The same process can be applied to skipped clusters between $C(\boldsymbol{u})$ and C_2 (see Fig. A3(b)). Consider the column of each such skipped cluster $\tilde{C}$. In this column, consider the cluster $\tilde{C}''$ that is in the same row as $C(\boldsymbol{s})$, and consider further the cluster $\tilde{C}'$ that is adjacent to $\tilde{C}''$ in the order for vertical traversal path extension (here downwards). Here, $\tilde{C}'$ is at the top of the column. The path established previously traversed $\tilde{C}'$ in the order $0, 3, 2, 1$. Now, instead of selecting the edge $3, 2$ in $\tilde{C}'$, we traverse the column of $\tilde{C}'$ (vertically, obviously) as follows. First, we exit $\tilde{C}'$ via procID 3. If the next cluster is $\tilde{C}$, then it is fully traversed according to $2, 1, 0, 3$, and subsequently exited. If the next cluster is not $\tilde{C}$, then it will be entered by the path twice: previously fully traversed according to $0, 3, 2, 1$, the original horizontal traversal now shrinks to $0, 1$ and the vertical traversal currently being established selects $2, 3$ and goes to the next cluster. Repeat this operation until returning to $\tilde{C}'$. In such a column, exactly one cluster ($\tilde{C}$) will be fully traversed by this vertical traversal for path extension, and all other clusters in the column will be entered twice by the path being constructed; see Fig. 13(b).

■

Lemma 9. *In a $TCC(k, 2)$, for $\boldsymbol{s} = (0, 0, 0)$ and a node $\boldsymbol{d} \in C(\boldsymbol{s})$ with procIDs of distinct parities, we can find a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$.*

Proof. We give a constructive proof. If $\boldsymbol{s} = \boldsymbol{d}$, this is the Hamiltonian cycle problem, and by Theorem 2 a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ can be found. Now, assume $\boldsymbol{s}, \boldsymbol{d}$ are distinct. Since $\boldsymbol{s} \neq \boldsymbol{d}$, $\boldsymbol{s}, \boldsymbol{d}$ are of distinct parities and since we assumed $\boldsymbol{s} = (0, 0, 0)$, either $p(\boldsymbol{d}) = 1$ or $p(\boldsymbol{d}) = 3$. If $p(\boldsymbol{d}) = 3$, we can apply the Hamiltonian cycle algorithm of Section 5, which will always select the edge $\boldsymbol{s} = (0, 0, 0) \rightarrow (0, 0, 3) = \boldsymbol{d}$. Effectively, the internal edges that are not selected in the Hamiltonian cycle are exclusively incident with nodes covering the same dimension. Discarding this edge $\boldsymbol{s} \rightarrow \boldsymbol{d}$ from the established Hamiltonian cycle induces a Hamiltonian path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$. If $p(\boldsymbol{d}) = 1$, the main idea is to consider clusters in the row of $C(\boldsymbol{s})$ as skipped clusters $\tilde{C}$ as in Step 9 of Lemma 8 the general case (with $\tilde{C}'$ clusters defined as in Step 9 of Lemma 8, that is, adjacent to $\tilde{C}$ in the order for vertical traversal path extension (here downwards)); see Fig. 14. ■

We summarise this discussion in Theorem 3 below.



(a) (b)

Fig. 14. Special case: $\boldsymbol{d} \in C(\boldsymbol{s})$ and $p(\boldsymbol{d}) = 1$. Path construction initialised in (a) and completed in (b).

Theorem 3. *In a $TCC(k, 2)$, we can find a Hamiltonian path between any two nodes of procIDs with distinct parities in $O(k^2)$ optimal time.*

Proof. A constructive proof is given in Lemmas 7 and 8. Regarding time complexity, one should note that all decision operations are made in constant time, for instance, by simply looking at the dimension of $\boldsymbol{d}$. The selection of one edge is also done in constant time. So, in order to select all the edges needed, the algorithm takes $O(k^2)$ time as we visit all the $4k^2$ nodes of a $TCC(k, 2)$. Obviously, this time complexity is optimal for a Hamiltonian path/lace. ■

7. Conclusions

In this paper, we have proposed a new topology for interconnection networks of massively parallel systems: torus-connected cycles. The aim of this topology is to mitigate the issues faced by most modern topologies described in the literature—issues which prevent actual hardware implementation of these topologies. Then, we have shown that the diameter of a $TCC(k, n)$ is nk in the case k is even, and nearly optimal in the case $k \geq 3$ is odd, precisely $nk + n$. We have deduced from this result a point-to-point routing algorithm that generates a path between any two nodes of length at most the diameter of the network. Next, we have shown the existence of and described an algorithm for constructing a Hamiltonian cycle inside a TCC. Finally, we have shown the existence of and proposed an algorithm for constructing a Hamiltonian path inside a two dimensional TCC.

Regarding future works, one may consider showing that the diameter of a $TCC(k, n)$ is nk for $k \geq 3$, where k is odd. Also, describing algorithms that solve the node-to-node and the node-to-set disjoint paths routing problems would be meaningful. Finally, fault-tolerance could be another possible research area.

Acknowledgment

This study was partly supported by a Grant-in-Aid for Scientific Research© of the Japan Society for the

Promotion of Science under Grant No. 25330079.

References

Al Faisal, F. and Rahman, M. (2009). Symmetric tori connected torus network, *Proceedings of the 12th International Conference on Computers and Information Technology (ICCIT), Dhaka, Bangladesh*, pp. 174–179.

Bossard, A. and Kaneko, K. (2012a). Node-to-set disjoint-path routing in hierarchical cubic networks, *The Computer Journal* **55**(12): 1440–1446.

Bossard, A. and Kaneko, K. (2012b). The set-to-set disjoint-path problem in perfect hierarchical hypercubes, *The Computer Journal* **55**(6): 769–775.

Bossard, A. and Kaneko, K. (2013). Set-to-set disjoint paths routing in hierarchical cubic networks, *The Computer Journal* **57**(2): 332–337.

Bossard, A., Kaneko, K. and Peng, S. (2010). Node-to-set disjoint paths routing in metacube, *Proceedings of the 22nd International Conference on Parallel and Distributed Computing and Systems (PDCS), Marina del Rey, CA, USA*, pp. 289–296.

Bossard, A., Kaneko, K. and Peng, S. (2011). A new node-to-set disjoint-path algorithm in perfect hierarchical hypercubes, *The Computer Journal* **54**(8): 1372–1381.

Camara, J.M., Moreto, M., Vallejo, E., Beivide, R., Miguel-Alonso, J., Martinez, C. and Navaridas, J. (2010). Twisted torus topologies for enhanced interconnection networks, *IEEE Transactions on Parallel and Distributed Systems* **21**(12): 1765–1778.

Duato, J., Yalamanchili, S. and Ni, L. (2003). *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann, San Francisco, CA.

Ghose, K. and Desai, K.R. (1995). Hierarchical cubic network, *IEEE Transactions on Parallel and Distributed Systems* **6**(4): 427–435.

Horiguchi, S. and Ooki, T. (2000). Hierarchical 3d-torus interconnection network, *Proceedings of the 5th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN), Dallas, TX, USA*, pp. 50–56.

Lai, C.-N. (2012). Optimal construction of all shortest node-disjoint paths in hypercubes with applications, *IEEE Transactions on Parallel and Distributed Systems* **23**(6): 1129–1134.

Li, Y., Peng, S. and Chu, W. (2004). Efficient collective communications in dual-cube, *The Journal of Supercomputing* **28**(1): 71–90.

Li, Y., Peng, S. and Chu, W. (2010). Metacube—a versatile family of interconnection networks for extremely large-scale supercomputers, *Journal of Supercomputing* **53**(2): 329–351.

Malluhi, Q.M. and Bayoumi, M.A. (1994). The hierarchical hypercube: A new interconnection topology for massively parallel systems, *IEEE Transactions on Parallel and Distributed Systems* **5**(1): 17–30.

Preparata, F.P. and Vuillemin, J. (1981). The cube-connected cycles: A versatile network for parallel computation, *Communications of the ACM* **24**(5): 300–309.

Seitz, C. (1985). The cosmic cube, *Communications of the ACM* **28**(1): 22–33.

Shih, Y.-K., Chuang, H.-C., Kao, S.-S. and Tan, J.J. (2010). Mutually independent Hamiltonian cycles in dual-cubes, *Journal of Supercomputing* **54**(2): 239–251.

Singh, A., Dally, W., Gupta, A. and Towles, B. (2003). Goal: A load-balanced adaptive routing algorithm for torus networks, *SIGARCH Computer Architecture News* **31**(2): 194–205.

TOP500 (2013). China's Tianhe-2 supercomputer takes no. 1 ranking on 41st TOP500 list, `http://top500.org/blog/lists/2013/06/press-release/`, (last accessed in August 2013).

Wu, J. and Sun, X.-H. (1994). Optimal cube-connected cube multicomputers, *Journal of Microcomputer Applications* **17**(2): 135–146.

Xiang, D. and Luo, W. (2012). An efficient adaptive deadlock-free routing algorithm for torus networks, *IEEE Transactions on Parallel and Distributed Systems* **23**(5): 800–808.

Zhou, S., Chen, L. and Xu, J. (2012a). Conditional fault diagnosability of dual-cubes, *International Journal of Foundations of Computer Science* **23**(8): 1729–1748.

Zhou, S., Lin, L. and Xu, J. (2012b). Conditional fault diagnosis of hierarchical hypercubes, *International Journal of Computer Mathematics* **89**(16): 2152–2164.

Antoine Bossard is an assistant professor of the Graduate School of Science, Kanagawa University, Japan. His research is focused on graph theory, interconnection networks and dependable systems. He received the B.E. and M.E. degrees from Université de Caen Basse-Normandie, France, in 2005 and 2007, respectively, and the Ph.D. degree from the Tokyo University of Agriculture and Technology, Japan, in 2011. He is a member of the ACM and ISCA.

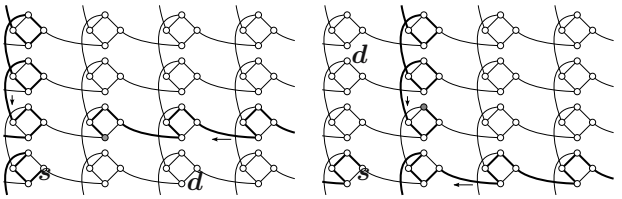
Keiichi Kaneko is a professor at the Tokyo University of Agriculture and Technology, Japan. His main research areas are dependable systems, interconnection networks, functional programming, parallel and distributed computation, partial evaluation and educational systems. He received the B.E., M.E. and Ph.D. degrees from the University of Tokyo in 1985, 1987 and 1994, respectively. He is a member of the IEEE, ACM, IEICE, IPSJ and JSSST.

Appendix A

d on the horizontal dimension: $\mathrm{C}(s)$ and $\mathrm{C}(d)$ share one dimension, the dimension of d

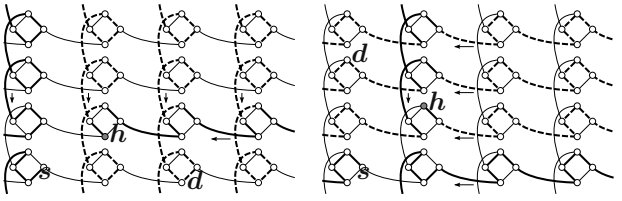
We are in the case $\boldsymbol{d}$ on the horizontal dimension. Now, if this dimension (horizontal dimension) is the one shared by

$C(\boldsymbol{s})$ and $C(\boldsymbol{d})$, then we need to slightly modify the initial path as given in Step 1 (see Fig. 10(b)) in order to avoid having $C(\boldsymbol{d})$, $C(\boldsymbol{u})$ and $C(\boldsymbol{h})$ in the same dimension (here the horizontal dimension). In practice, we shall initialise the path such that the head of the path is shifted to another row. Details are given in Fig. A1(a). Then, the path is extended as detailed in the general case (cf. Lemma 7); see Fig. A2(a). Finally, similar to the general case, C_1 and C_2 are positioned, and routing is performed accordingly; see Fig. A3(a). Skipped clusters $\tilde{C}$ are processed similarly as in the general case, with $\tilde{C}'$ in the row of $\tilde{C}$ and the column of $C(\boldsymbol{s})$.



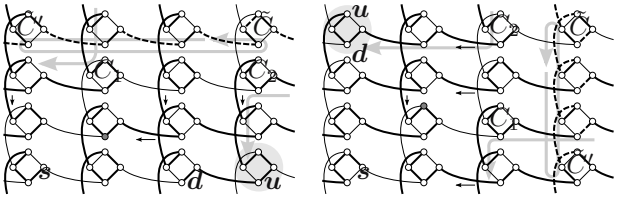
(a) (b)

Fig. A1. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, the dimension of $\boldsymbol{d}$. Path initialisation for vertical (a) and horizontal (b) traversals. Newly selected edges in bold; the greyed node is the head of the (still in construction) path.



(a) (b)

Fig. A2. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, the dimension of $\boldsymbol{d}$. Path extension for vertical (a) and horizontal (b) traversals. Newly selected edges in dashes.



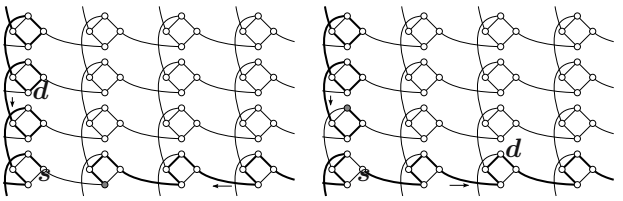
(a) (b)

Fig. A3. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, the dimension of $\boldsymbol{d}$. Extending the path to include skipped clusters $\tilde{C}$ (if any) in the case of vertical (a) and horizontal (b) traversals. Path extension emphasised by dashed edges.

Appendix B

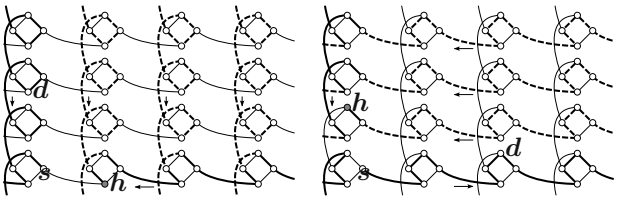
d on the horizontal dimension: $C(s)$ and $C(d)$ share one dimension, but not the dimension of d

Now, we are in the case $\boldsymbol{d}$ on the horizontal dimension, and $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share the vertical dimension. The algorithm described in Lemma 7 for the general case can be applied with the exception that there is no C_1 needed: $C(\boldsymbol{u})$ can be reached directly from $C(\boldsymbol{h})$ without elbow (so, straight). Successive steps are illustrated in Figs. B1(a), B2(a) and B3(a). Skipped clusters $\tilde{C}$ are processed similarly as in the general case, with $\tilde{C}'$ in the row of $\tilde{C}$ and the column of $C(\boldsymbol{s})$.



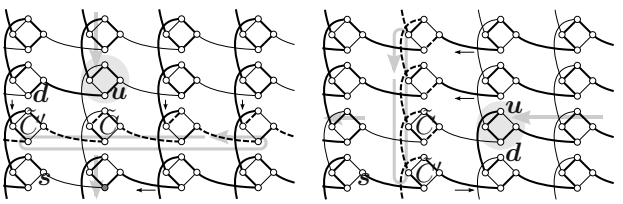
(a) (b)

Fig. B1. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, but not the dimension of $\boldsymbol{d}$. Path initialisation for vertical (a) and horizontal (b) traversals. Newly selected edges in bold; the greyed node is the head of the (still in construction) path.



(a) (b)

Fig. B2. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, but not the dimension of $\boldsymbol{d}$. Path extension for vertical (a) and horizontal (b) traversals. Newly selected edges in dashes.



(a) (b)

Fig. B3. Special case: $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share one dimension, but not the dimension of $\boldsymbol{d}$. Extending the path to include skipped clusters $\tilde{C}$ (if any) in the case of vertical (a) and horizontal (b) traversals. Path extension emphasised by dashed edges.

Appendix C

d on the vertical dimension: $C(s)$ and $C(d)$ share one dimension, the dimension of d

We are in the case $\boldsymbol{d}$ on the vertical dimension. Now, if this dimension (vertical dimension) is the one shared by $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$, then we need to slightly modify the initial path as given in Step 1 (see Fig. 10(b)) in order to avoid having $C(\boldsymbol{d})$, $C(\boldsymbol{u})$ and $C(\boldsymbol{h})$ in the same dimension (here the vertical dimension). In practice, we shall initialise the path such that the head of the path is shifted to another column. Details are given in Fig. A1(b). Then, the path is extended as detailed in the general case (cf. Lemma 8); see Fig. A2(b). Finally, similar to the general case, C_1 and C_2 are positioned, and routing is performed accordingly; see Fig. A3(b). Skipped clusters are processed similarly as in the general case with the exception that, considering the column of each such skipped cluster $\tilde{C}$, we define the cluster $\tilde{C}'$ of this column to be in the same row as $C(\boldsymbol{s})$.

Appendix D

d on the vertical dimension: $C(s)$ and $C(d)$ share one dimension, but not the dimension of d

Now, we are in the case $\boldsymbol{d}$ on the vertical dimension and $C(\boldsymbol{s})$ and $C(\boldsymbol{d})$ share the horizontal dimension. The algorithm described in Lemma 8 for the general case can be applied with the exception that there is no C_1 needed: $C(\boldsymbol{u})$ can be reached directly from $C(\boldsymbol{h})$ without elbow (so, straight). Skipped clusters are processed similarly as in the general case with the exception that, considering the column of each such skipped cluster $\tilde{C}$, we define the cluster $\tilde{C}'$ of this column to be in the same row as $C(\boldsymbol{s})$.

Received: 9 April 2014
Revised: 28 October 2014