IEICE TRANS. INF. & SYST., VOL.E100–D, NO.8 AUGUST 2017

PAPER

A Novel Channel Assignment Method to Ensure Deadlock-Freedom for Deterministic Routing

Ryuta KAWANO[†a)], ***Student Member***, **Hiroshi NAKAHARA**[†], **Seiichi TADE**[†], ***Nonmembers***, **Ikki FUJIWARA**[††], **Hiroki MATSUTANI**[†], ***Members***, **Michihiro KOIBUCHI**[†††], ***Senior Member***, ***and*** **Hideharu AMANO**[†], ***Fellow***

SUMMARY Inter-switch networks for HPC systems and data-centers can be improved by applying random shortcut topologies with a reduced number of hops. With minimal routing in such networks; however, deadlock-freedom is not guaranteed. Multiple Virtual Channels (VCs) are efficiently used to avoid this problem. However, previous works do not provide good trade-offs between the number of required VCs and the time and memory complexities of an algorithm. In this work, a novel and fast algorithm, named ACRO, is proposed to endorse the arbitrary routing functions with deadlock-freedom, as well as consuming a small number of VCs. A heuristic approach to reduce VCs is achieved with a hash table, which improves the scalability of the algorithm compared with our previous work. Moreover, experimental results show that ACRO can reduce the average number of VCs by up to 63% when compared with a conventional algorithm that has the same time complexity. Furthermore, ACRO reduces the time complexity by a factor of $O(|N| \cdot \log |N|)$, when compared with another conventional algorithm that requires almost the same number of VCs.

key words: *deadlock-free routing, high performance computing, time complexity*

1. Introduction

For large parallel applications executed on the next generation of High Performance Computing (HPC) systems, MPI communication latency should be lower than one microsecond [1], [2]. In order to cope with this requirement, low-latency inter-switch networks are needed. Switch delays (e.g., about 100 nanoseconds in InfiniBand QDR) are typically larger than the wire and flit injection delays, even when including serial and parallel converters. Therefore, to achieve low latency, inter-switch topologies should have low diameter and low average shortest path length, both of which can be measured in terms of number of switch hops. Compared with the conventional Torus or Fat-tree networks, recently proposed random shortcut topologies can drastically reduce the number of hops [3]–[5]. It is reported that such topologies can be efficiently applied to inter-switch networks for HPC systems and data-centers. To achieve low latency networks with irregular topologies, optimized routing methods including minimal routing should be utilized. Most of these routing methods do not guarantee deadlock-freedom that has to be supported in practical HPC networks. Therefore, endorsing a given livelock-free routing, including minimal routing methods, with deadlock-freedom is needed.

In this work, multiple Virtual Channels (VCs) [6], [7] for each physical channel are exploited in order to support deadlock-freedom for arbitrary routing methods. The proposed methodology takes a given topology and the routing table, obtained from a livelock-free routing algorithm, as inputs to generate the VC assignment to paths. This approach has a small time complexity, yet with the same number of VCs when compared with conventional methods.

This work is based on our previous research [8] which suffers from the lack of scalability in terms of network size. In order to improve it, a new algorithm named Assignment of Channels in Reverse Order (ACRO) using a hash table is introduced, and detailed evaluation results are shown.

The rest of the paper is organized as follows. Section 2 shows the related work. In Sect. 3, the detailed ACRO algorithm is presented. In Sect. 4, the proposed algorithm is evaluated and compared with conventional VC assignment methods. Finally, we conclude the paper in Sect. 5.

2. Related Work

2.1 Low-Latency Random Topologies for HPC Systems

End-to-end latencies can be reduced by using networks with high-radix switches. Such high-radix networks include Flattened Butterfly [9] and Dragonfly [10] which are utilized in the latest supercomputers. However, they necessitate several long links between cabinets that increase the cost. K-ary n-Torus with a large K used in Blue Gene/Q [11] and K-supercomputer [12] is another choice of low latency networks. However, using large Ks also increases the number of long links.

Researchers in the field of applied graph theory have focused on the optimum Moore graphs [13]. The number of nodes in these graphs can reach the Moore bound with given degree and diameter values. As for networks for high-performance computing systems, diameter-2 optimum or quasi-optimum graphs are adopted. Hoffman-Singleton graph [14], which is one of the well-known Moore graphs, is utilized to optimize networks among servers in

Manuscript received December 1, 2016.
Manuscript revised March 28, 2017.
Manuscript publicized May 19, 2017.

[†]The authors are with Dept. of ICS, Keio University, Yokohama-shi, 223–0061 Japan.
[††]The author is with Universal Communication Research Institute, National Institute of Information and Communications Technology, Kyoto-shi, 619–0289 Japan.
[†††]The author is with National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: blackbus@am.ics.keio.ac.jp

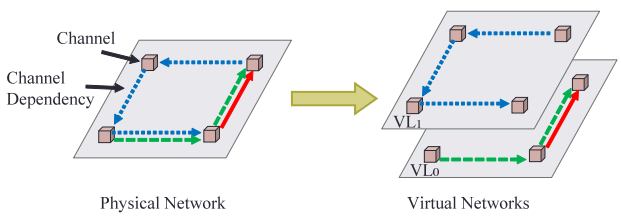
DOI: 10.1587/transinf.2016EDP7477

a cabinet [15]. Moreover, quasi-optimum McKay-Miller-Širáň (MMS) graph [16] can be adopted for Slim Fly topology [17], which realizes diameter-2 large-scale networks among top-of-rack (ToR) switches. However, the sizes and degrees of such a network are strictly limited, and complicated heuristics are often needed to find the required network configuration [18].

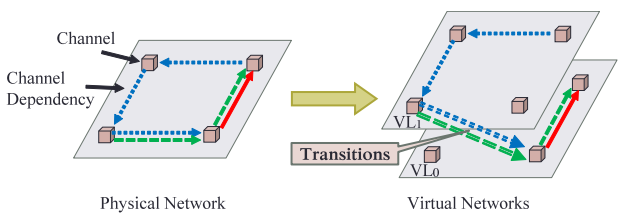
Random topologies for HPC systems can achieve latencies as low as optimized topologies, yet they have to be generated with much simpler algorithms. The latency reduction achieved by the small-world phenomenon is proportional to $\log_d |N|$, where d and $|N|$ represent the degree of each node and the number of nodes, respectively. They can improve the bandwidth, scalability, and fault-tolerance of inter-ToR networks [3]–[5] and inter-router on-chip networks [19].

2.2 Methodologies for Designing Deadlock-Free Routing

In this work, we cannot use minimal adaptive routing with escape paths [20] to support deadlock-freedom since routing tables are assumed to be given. Alternatively, in this work, multiple Virtual Channels (VCs) are exploited to break cyclic channel dependencies, as used in LASH [6], [21] and LASH-TOR [7] routings. Both of these two conventional routing techniques can assign each path between source and destination nodes to Virtual Layers (VLs), each of which is constructed with one of the VCs. The difference between them is whether to permit transitions among VLs for each path. In LASH routing, paths' transitions never occur, and each path must be assigned to one of the VLs, as shown in Fig. 1 (a). It supports deadlock-free minimal paths at the cost of a relatively large number of VLs. For instance, it uses



(a) LASH routing.



(b) LASH-TOR routing.

Fig. 1 Routings with virtual layers.

Table 1 Trade-offs on conventional algorithms.

Routing	# of required VLs	Time Complexity
LASH	9 for 256 SWs [7]	$O(\|N\|^2)$
LASH-TOR	4 for 256 SWs [7]	$O(\|N\|^3)$

up to 9 VLs for the 256-nodes case [7]. In LASH-TOR routing, on the other hand, each path can be split into sub-paths among ordered VLs. Thus, it can achieve deadlock-freedom with only 4 VLs for the same 256-nodes case [7]. However, the time complexity can reach up to $O(|N|^3)$, which makes it quite unrealistic for large-sized networks. The trade-offs between the two routing methods are summarized in Table 1.

To avoid cyclic channel dependencies in each VL, we introduce ordered channels [22] that determine the following theorem.

Theorem 1. *A set of paths within a network is deadlock-free if, and only if, there exists an channels' ordering such that each path uses the channels in a decreasing order.*

This can be proved by a topological sort for a Channel Dependency Graph (CDG) induced by a set of paths [23]. We propose a heuristic approach that aims to construct the channel ordering in each VL in order to maximize the number of paths and the length of each path routed within the VL. The number of required VLs is minimized with this heuristic approach to use the transitions among the VLs mentioned above.

3. ACRO for Deadlock-Free Routing

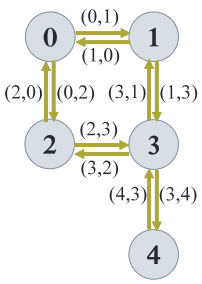
3.1 Problem Definition

According to the general models [22], the configuration of an interconnection network is defined as follows.

Definition 1. *An interconnection network I is represented by a directed graph $I = (N, C)$, where N is a set of switches and C is a set of physical channels.*

Definition 2. *A deterministic routing function $R : N \times N \rightarrow C$ returns the physical output channel c_{out} to be taken from a node n_{i} for packets whose destination node is n_{d}.*

In our methodology, a topology and a routing table, represented in Fig. 2, are given as an interconnection network and a routing function, respectively. The routing table in Fig. 2 (b) takes a current node n_{i} and a destination node n_{d} as inputs, and returns the next node n_{next}. Therefore, the output channel c_{out} in Def. 2 is determined as follows: c_{out} is $(n_{\text{i}}, n_{\text{next}})$ if $n_{\text{i}} \neq n_{\text{d}}$; otherwise, c_{out} becomes $c_{\text{local},n_{\text{d}}}$.



(a) Topology.

S \ D	0	1	2	3	4
0		1	2	1	1
1	0		3	3	3
2	0	0		3	3
3	2	1	2		4
4	3	3	3	3	

(b) Routing Table.

Fig. 2 An example of given inputs.

The definition of a VL L_i in a network I is the same as that adopted in LASH-TOR [7]; that is, L_i can be treated as a virtual network that is isomorphic to the original physical network I. Additionally, in this work, L_i is defined as a strictly and decreasingly ordered set. It contains sorted physical channels of I such that every path within L_i is restricted to use the corresponding VCs in a decreasing order. Namely, each VL is a sorted set of C and is denoted as $L_i := \{c_{i,|C|-1}, \ldots, c_{i,0}\}$.

Given the inputs of a topology and a routing table, a strictly and decreasingly ordered set of layers $L = \{L_{|L|-1}, \ldots, L_0\}$ is determined such that for every (source, destination) pair, the path can reach the destination by using channels in decreasing order within each L_i and transitions among layers in a decreasing order within L.

3.2 CDG (Channel Dependency Graph) Generation for Each Destination

In the conventional implementation of LASH-TOR [7], at least a cyclic dependency check must be done for a path. Since a cyclic dependency search has a time complexity of $O(|C| + |E|)$, the minimum time complexity per VL becomes approximately $O(|N|^3)$. In the recent improvement on LASH [21], only a cyclic dependency check per VL is needed. This can be done by initially adding all paths to a CDG and checking the cyclic dependencies for all edges. Detected cycles are removed by moving a portion of paths included in each cycle to the next VL. Then, the dependency check is done again. It is iterated until no cyclic dependency is detected. This solution is called an offline-manner which can reduce the time complexity of search to $O(|N|^2)$. Here, the extension of this offline-manner for LASH-TOR, called ACRO is proposed to balance the number of VLs and the time complexity.

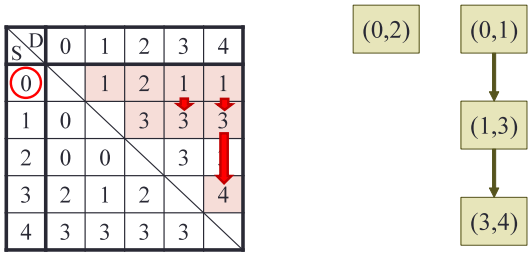
The details of the proposed ACRO algorithm are shown in Alg. 1. A set of paths from a set of nodes N to a destination node n_d is determined as $T'_{n_\mathrm{d}} = (N, C_{n_\mathrm{d}})$, satisfying $C_{n_\mathrm{d}} \subset C$. T'_{n_d} forms a directed tree whose root is n_d, and all edges are directed toward the direction of n_d. T'_{n_d} produces a CDG for the destination n_d. Here, it is represented as $T_{n_\mathrm{d}} = (C, E_{n_\mathrm{d}})$, where E_{n_d} denotes a set of the channel dependencies. T_{n_d} is a set of directed trees, as shown in Fig. 4 (b), and the node of the CDG is corresponding to 'channel.' To avoid any confusion, a node in a CDG is called a 'channel.'

In this work, all of the channel dependencies with all paths in a traffic are determined as a set of CDGs for the destination nodes rather than the source nodes. This choice can be explained by the fact that, and as shown in Fig. 3 (b), if a CDG is generated for each source node, the number of referred elements in the table for each destination node is equal to the number of hops between the source and destination nodes (Fig. 3 (a)). Therefore, the time complexity becomes $O(|N| \cdot \log |N|)$ for each source node, where $O(\log |N|)$ comes from the characteristics of irregular networks [24]. On the other hand, and as depicted in Fig. 4 (b), if a CDG is generated for each destination node, only a column of the table is needed to be referred to (Fig. 4 (a)). As a result, the time complexity becomes only $O(|N|)$ for each destination node.

Algorithm 1 Assignment of Virtual Layers.

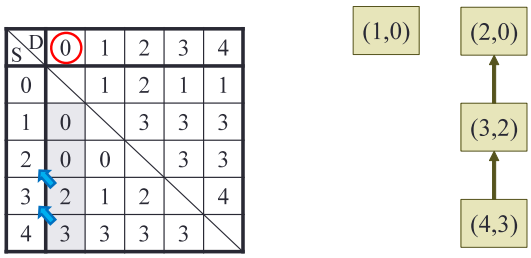
Input: $I = (N, C)$, a Routing Table
Output: a Set of Virtual Layers L

```
for all n ∈ N do
    Create a CDG T_n = (C, E_n)
end for

for all n ∈ N do
    Calculate sets of weight values;
    {(h_{n,c}, w_{n,c}) | c ∈ C} (See Sect. 3.3.1)
end for

for all c ∈ C do
    Create an empty hash table H_c
    for 0 ≤ Δ < D do
        H_c[Δ] ← 0
    end for
end for
/* Initially calculate fitness values */
for all n ∈ N do
    for all c ∈ C do
        if c has any parent in T_n then
            H_c[h_{n,c}] ← H_c[h_{n,c}] + w_{n,c}
        end if
    end for
end for

for all c ∈ C do
    f(c) ← max({Δ | 0 ≤ Δ < D, H_c[Δ] > 0} ∪ {0})
end for

Set boolean values {m(n, c) = False | n ∈ N, c ∈ C}

/* Set Virtual Layers repeatedly */
i ← 0
repeat
    Create a new empty strictly ordered set L_i
    Create a set U, a copy set of C
    while U ≠ ∅ do
        From U remove u_min which minimizes f(u),
        breaking ties by H_u[f(u)]
        Add u_min to the head of L_i
        for all n ∈ N do
            if u_min has no parent in T_n then
                m(n, u_min) ← True
                for all c' ∈ C_child(n, u_min) do
                    Remove an edge (c', u_min) in T_n
                    H_{c'}[h_{n,c'}] ← H_{c'}[h_{n,c'}] − w_{n,c'}
                    while H_{c'}[f(c')] = 0 ∧ f(c') > 0 do
                        f(c') ← f(c') − 1
                    end while
                end for
            end if
        end for
    end while
    Add L_i to the head of L
    i ← i + 1
until ∀n ∈ N ∀c ∈ C, m(n, c) = True
```



(a) Routing Table. (b) CDG for src. #0.

Fig. 3 Creation of CDG for src. #0.



(a) Routing Table. (b) CDG for dst. #0.

Fig. 4 Creation of CDG for dst. #0.

3.3 Heuristic Approach to Reduce VLs

3.3.1 Weight Values of Channel in CDG

VCs in each VL should be ordered properly to minimize the number of required VLs for deadlock-freedom. In this section, a simple heuristic approach is introduced to minimize the number of paths and the length of each path moved to the next VL.

$C_{\text{child}}(n, c)$ is defined as a set of channels which are children of the channel c in a CDG T_n. Two weight values, $h_{n,c}$ and $w_{n,c}$, are introduced for c. $h_{n,c}$ represents the "height" of c, and $w_{n,c}$ represents the number of the "deepest" leaves of c. These values are calculated by the following equations:

$$h_{n,c} = \begin{cases} 0 & (\text{if } C_{\text{child}}(n,c) = \emptyset) \\ 1 + \max H_{\text{child}}(n,c) & (\text{otherwise}) \end{cases}$$

where

$$H_{\text{child}}(n,c) := \{h_{n,c'} | c' \in C_{\text{child}}(n,c)\}$$

and

$$w_{n,c} = \begin{cases} 1 & (\text{if } C_{\text{child}}(n,c) = \emptyset) \\ \displaystyle\sum_{c' \in C'(n,c)} w_{n,c'} & (\text{otherwise}) \end{cases}$$

where

$$C'(n,c) := \{c' | c' \in C_{\text{child}}(n,c), h_{n,c'} = \max H_{\text{child}}(n,c)\}$$

3.3.2 Hash Table for Each Channel

After introducing the weight values of a channel in a given CDG, it has to be determined which channel of the cycle to be broken in order to minimize the length and the number of paths moved to the next VL. Consequently, this leads to minimize the number of VLs needed. In the proposed algorithm, a hash table for each channel c, $\mathbb{H}_c$, is introduced. It manages the number and length of the longest paths that would be moved to the next VL by the channel. Note that the longest paths are adopted as objectives, rather than all paths including shorter paths.

In the earlier stage of this research [8], a fitness value $f(c)$ was used; however, it has to be implemented with a long integer number that requires a large overhead for large network sizes. The effect of this substitution is evaluated in Sect. 4.1.2.

Considering all paths would make the memory complexity in the algorithm quite large. Moreover, the number of dependencies to be solved is dominant to the longest paths, and shorter paths are negligible.

The values of the tables are initially calculated as follows. If a channel c has any parent in T_n, a value in $\mathbb{H}_c$ corresponding to the key $h_{n,c}$ is incremented by $w_{n,c}$. Note that this increment is not applied in the case of a root channel c because if the smallest order is set to this root channel, no path would be moved to the next VL.

3.4 Assignment of VLs to Paths in Reverse Order

To make sure whether a channel is reachable in T_n, an $|N| \times |C|$ boolean table is introduced. Each element of the table is initially set to a boolean value 'False', which is specified by $m(n, c)$ in Alg. 1. The termination condition of the algorithm is that all channels $c \in C$ are reachable in T_n for all destinations $n \in N$.

The VLs and VCs are assigned for each path in the reverse order. Namely, unlike the conventional virtually layered networks, the VLs and VCs are assigned in the order from the destination node to the source node.

After generating a new VL L_i as an empty ordered set, the order of channels is fixed in a one-by-one fashion. Among the channels whose orders are not assigned yet, a channel $u_{\min}$, which minimizes the largest key with a non-zero value in H_u, $f(u)$, is selected to append to the head of L_i. If there are some ties, they are broken by selecting one of them which minimizes the value of $\mathbb{H}_u[f(u)]$.

For each CDG T_n, if the channel $u_{\min}$ does not have a parent in T_n, the following procedures are performed. Since the packet whose destination is n can reach $u_{\min}$ in the current VL L_i, the boolean value of 'True' is assigned to $m(n, u_{\min})$. Furthermore, edges from all the children of $u_{\min}$ to $u_{\min}$ itself are removed if they exist.

The deletion of the edges denotes that the dependency between the child of $u_{\min}$ and $u_{\min}$ will be dissolved in either of the following two ways. If the order of the child is not as-

signed yet, it will be inevitably larger than that of $u_{\min}$. This means that the dependency is dissolved within the current L_i. Otherwise, the order of the child is surely smaller than that of $u_{\min}$, which cannot dissolve the dependency. Even after all the channel orders are assigned in L_i, the termination condition is not satisfied. This leads to the generation of a VL L_{i+1}. The dependency will be dissolved by the transition between the child in L_{i+1} and $u_{\min}$ in L_i.

After its execution, the deletion is reflected to the values of the hash tables by the decrement of $\mathbb{H}_{c'}[h_{n,c'}]$ by $w_{n,c'}$ for each child of $u_{\min}$, $c' \in C_{\text{child}}(n, u_{\min})$.

3.5 Deadlock-Free Routing with ACRO

3.5.1 Implementation on Switches

A possible implementation of ACRO is that each switch has a mapping table for VLs. The table holds boolean values which represent whether an order of each input channel is larger than that of each output channel.

Packets are restricted to be injected to an output channel of the first hop with a VL $L_{|L|-1}$, which has the maximum order among all VLs. In the subsequent hops, the table entries mentioned above are referred to in each switch. If the input channel has larger order than the output channel within a VL L_i that is used in the input, the same VL L_i is used in the output; otherwise, L_{i-1} is used. It is important to mention that the number of table entries for each switch is independent of the number of switches $|N|$; but, it depends on the number of input and output channels and the number of required VLs. This reduced amount of information for VL mapping enables a scalable implementation for larger system sizes.

3.5.2 Livelock- and Deadlock-Freedom with ACRO

Livelock- and deadlock-freedom supported by ACRO is proved by the following two theorems:

Theorem 2. *A path exists between an arbitrary source-and-destination pair with ACRO.*

Proof. *With the proposed algorithm described in Sect. 1, a path between an arbitrary source-and-destination pair is contained in a strictly ordered set of layers* $L' = \{L_{|L'|-1}, \ldots, L_0\}$. *When* $|L'| > 1$ *is satisfied, transitions between layers are performed in the switches given by the strictly ordered set* $N' = \{n_{|L'|-2}, \ldots, n_0\}$. *Deadlock-freedom among VLs is supported with this path division. Moreover, each subpath in each VL uses the channels in a strictly decreasing order, which supports deadlock-freedom within each VL.* □

Theorem 3. *The VC and VL assignment with ACRO and its implementation described in Sect. 3.5.1 can endorse a given topology and a routing table with both livelock- and deadlock-freedom.*

Proof. *Each VC belonging to c in a VL* L_i *is labeled with a two-digit identifier* $(i, \text{Idx}(i, c))_{|C|}$, *where* $Idx(i, c)$ *is the order of c in* L_i, *and* $(i, j)_{|C|} = i \cdot |C| + j$. *Given these labeling to VCs, a packet in each switch with the implementation in Sect. 3.5.1 uses a pair of input and output channels which are labeled in strictly decreasing order. Since all packets are initially injected to the VL with the maximum order, VCs traversed by the packets always have labels equal to or larger than those in the same channel containing the established paths in Theorem 2; therefore, packets are transferred with VCs whose labels are in a strictly descending order until reaching the destination nodes.* □

4. Evaluation

In this section, the proposed VC assignment method is evaluated and compared with the conventional VC assignment methods proposed in LASH and LASH-TOR. As shown in Sect. 3, a topology of switches and a routing table are given as inputs. Note that the routing table takes source and destination nodes, and returns the next node.

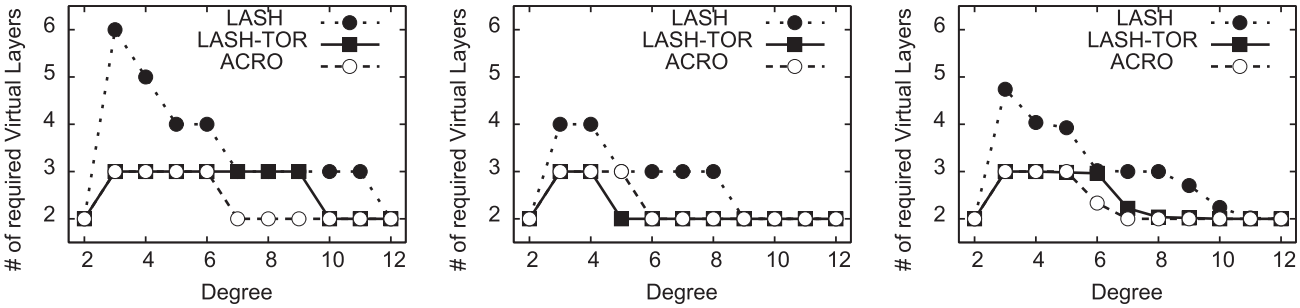
4.1 Number of Required VLs

In this section, the impact of the network size and the node degree to the number of VLs is analyzed. Here, the degree is corresponding to the number of ports of a switch.

4.1.1 Implementation of VC Assignment Algorithm

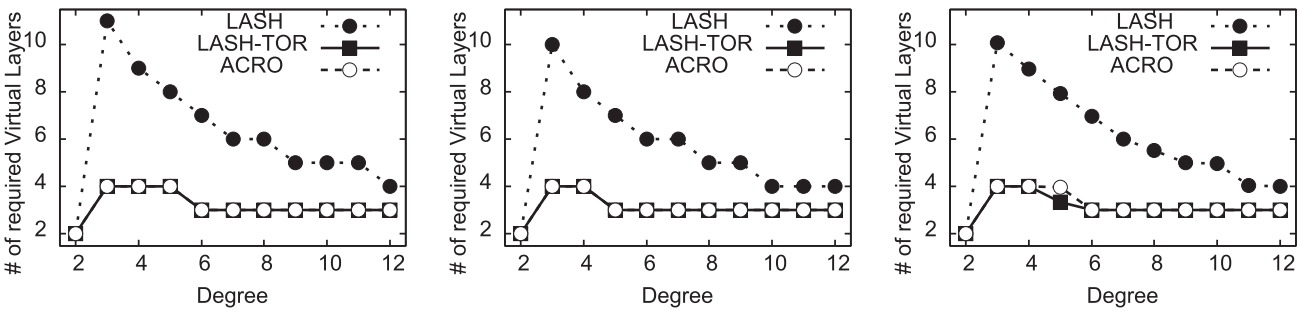
In this evaluation, the VC assignment algorithm in LASH is implemented as follows. Let $\text{G}_{\text{CD},i}$ be a CDG created by a set of paths in L_i. For a given path between a source node n_s and a destination node n_d, the algorithm searches a set of VLs L to find $L_i \in L$. The path induces the channel dependencies, which could be added to $\text{G}_{\text{CD},i}$ without generating a cycle of channel dependencies. If L_i is found, the dependencies are added to $\text{G}_{\text{CD},i}$. Otherwise, a new VL and the corresponding CDG are created and the channel dependencies are added to the new CDG.

The VC assignment algorithm in LASH-TOR is implemented in a similar way as that in LASH: let G_{CD} be a set of CDGs created by sets of paths in a set of VLs L. For the same inputs as in LASH, the algorithm searches L and N to find $\{L', S\}$, satisfying $L' \subseteq L$ and $S \subset N$. The path would be split into the subpaths according to the transition node $s_j \in S$. Each subpath would induce the channel dependencies, which could be added to each $\text{G}_{\text{CD},i} \in \text{G}_{\text{CD}}$ without generating a cycle of dependencies within each $L_i \in L'$ respectively. If $\{L', S\}$ is found, the dependencies are added to G_{CD}; otherwise, a new VL and the corresponding CDG are created and added to L and G_{CD}, respectively. G_{CD} then incorporates the dependencies obtained from $\{L', S\}$ that satisfies the condition mentioned above, and is exhibited by the additional VL. The original implementation of LASH-TOR can limit the number of VLs by permitting non-minimal paths with up*/down* routing on the final VL. On



(a) Maximum. (b) Minimum. (c) Average.

Fig. 5 Number of required VLs (64 nodes).



(a) Maximum. (b) Minimum. (c) Average.

Fig. 6 Number of required VLs (256 nodes).

the other hand, the proposed method ACRO completely supports paths induced by a given routing table. Thus, it does not use the alternative longer paths. For fairness of comparison, the number of maximum VLs are not limited in LASH-TOR; therefore, up*/down* routing is not used in the last VL during this evaluation.

4.1.2 Experimental Results

Surely connected regular random topologies are adopted in this evaluation, in which all of the edges are bidirectional. The number of nodes is set to $|N| = 64$ and 256. The number of degree d is varied from 4 to 12. In this evaluation, a hundred topologies are generated from different seeds for each $(|N|, d)$ pair. The corresponding routing tables take exactly one minimal path for a source-and-destination pair.

Figure 5 and Fig. 6 show the maximum, minimum, and average numbers of required VLs for 64- and 256-node topologies, respectively. These results show that ACRO efficiently reduces the number of VLs compared with the VC assignment methodology in LASH routing. For 64-node topologies, it reduces the average and maximum numbers of required VLs by up to 37% and 50%, respectively. Furthermore, for 256-node topologies, it reduces the average and maximum numbers of required VLs by up to 60% and 63%, respectively. Another interesting result is that the heuristic used in ACRO achieves almost the same number of required VLs as the VC assignment methodology in LASH-TOR routing in which paths are assigned to VLs sequentially.

Moreover, ACRO accomplishes as small variance in the number of required VLs as LASH-TOR. In our evaluation, a difference of 2 between the maximum and the minimum numbers of required VLs is observed for LASH in the case of $(|N|, d) = (64, 3)$. On the other hand, the differences for ACRO and LASH-TOR never exceed 1.

As shown in Sect. 3.3.2, a fitness value was utilized in our earlier stage of research [8]. It was implemented as a long integer whose length was more than 1,000 bits for 256 nodes. In this work, contrarily, it is replaced with a hash table to improve the scalability of the algorithm; thus, it can be implemented as a typical 32-bit integer. However, this change had negligible influence on the number of required VLs.

4.2 Time and Memory Complexity

The original algorithm for the VC assignment in LASH [6] is accelerated by a recent implementation [21]. In this implementation, the time complexity and the memory complexity of VC assignment algorithm are as follows:

Proposition 1. *The time complexity of the algorithm for VC assignment in LASH is*

$$O(\nabla \cdot (|C| + |E|) + |N|^2)$$

while the memory complexity is

$$O(\nabla \cdot D(I) \cdot |N|^2 + \nabla \cdot (|C| + |E|)).$$

In this proposition, ∇ and $D(I)$ are a number of required

VLs and the diameter of a topology, respectively. Parameters in the above proposition are to be hereinafter used.

On the other hand, the time and memory complexities of VC assignment algorithm in LASH-TOR [7] are as follows:

Proposition 2. *The time complexity of the algorithm for VC assignment in LASH-TOR is*

$$O(|N|^2 \cdot D(I) \cdot \nabla \cdot (|C| + |E|))$$

while the memory complexity is

$$O(\nabla \cdot (|C| + |E|))$$

From the proposed ACRO algorithm shown in Alg. 1, the time and memory complexities are summarized as follows. The generation of CDGs for all the destinations has a time complexity of $O(|E| \cdot |N|)$ and a memory complexity of $O(|C| \cdot |N|)$. A time complexity to calculate all weight values $h_{n,c}, w_{n,c}$ is $O(|E| \cdot |N|)$, while the memory complexity is $O(|C| \cdot |N|)$. Initialization of the values in hashes $\mathbb{H}_c$ has a time complexity of $O(|C| \cdot |N|)$ and a memory complexity of $O(|C| \cdot D(I))$. Checking whether $u_{\min}$ has any parent in T_n needs a time complexity of $O(\nabla \cdot |C| \cdot |N|)$ in total. The selection of the channel which minimizes the fitness function $f(u)$ has a time complexity of $O(|C|^2 \cdot \nabla)$ in total. Moreover, the modification of $\mathbb{H}_{c'}$ and $f(c')$ needs time complexities of $O(\nabla \cdot |C| \cdot |N|)$ and $O(\nabla \cdot |C| \cdot D(I))$ in total, respectively.

The findings mentioned above are followed by these propositions:

Proposition 3. *The time complexity of the ACRO algorithm is*

$$O(\nabla \cdot |C| \cdot (|N| + |C| + D(I)) + |N| \cdot |E|)$$

while the memory complexity is

$$O(|C| \cdot (|N| + D(I)).$$

Given that a topology is randomly generated and the degree d is quite smaller than the network size $|N|$, the proportionalities $D(I) \propto \log |N|$, $|C| \propto |N|$, and $|E| \propto |N|$ are satisfied [24]. From these proportionalities, it can be said that ACRO reduces the time complexity by a factor of $O(|N| \cdot \log |N|)$ when compared with that of the VC assignment algorithm in LASH-TOR, yet with almost the same number of required VLs.

4.3 Algorithm Execution Time

In this section, execution time of ACRO itself is compared with that of the conventional VC assignment methods; LASH and LASH-TOR described in Sect. 4.1.1. All methods were implemented with Python scripts that use a NetworkX package. They are executed on a server with two Intel Xeon CPUs E5-2470 @ 2.30 GHz (2x8 cores) and 128GB memory. The number of nodes is varied among $|N| = 64, 256$ and 1024, and the number of degree d is set to 16. Ten topologies are generated from different seeds for each node.

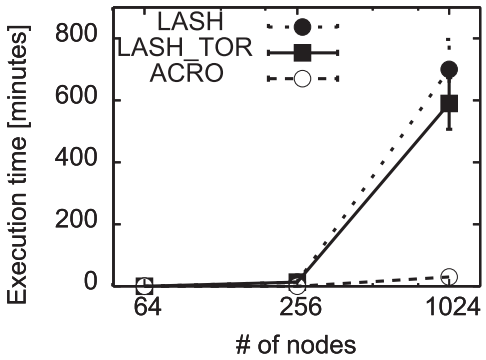


Fig. 7 Execution time of each algorithm ($d = 16$).

The average and standard deviation values are shown in Fig. 7. The results show that ACRO completed its VC assignment in about 30 minutes for 1,024 nodes. On the other hand, LASH and LASH-TOR consumed about 589 and 700 minutes for the same network size, respectively. In this experiment, ACRO was executed 2.3, 8.4, and 19.9 times faster than LASH-TOR for 64, 256, and 1,024 nodes, respectively. These results demonstrate the small time complexity of ACRO, previously mentioned in Sect. 4.2.

5. Conclusion

In this work, a novel algorithm to make arbitrary routing methods for irregular networks deadlock-free was proposed with a reasonable number of virtual channels and a time complexity. In the algorithm, VCs are assigned to paths from the destination node to the source node. In order to remove cyclic dependencies with a number of VLs as small as possible, a heuristic approach is introduced. The proposed VC assignment algorithm, ACRO, has a quite small time complexity yet requires almost the same number of VLs compared with that in LASH-TOR. Experimental results show that ACRO supports both the reduced number of VLs and the time complexity as small as LASH at the same time.

Acknowledgments

A part of this work was supported by JSPS KAKENHI Grant Number JP 15J03374.

References

[1] K. Scott Hemmert et al, "Report on Institute for Advanced Architectures and Algorithms," Interconnection Networks Workshop 2008, http://ft.ornl.gov/doku/_media/iaaicw/iaa-ic-2008-workshop-report-v09.pdf

[2] J. Tomkins, "Interconnects: A Buyers Point of View," ACS Workshop, June 2007.

[3] J.Y. Shin, B. Wong, and E.G. Sirer, "Small-World Datacenters," Proc. Symposium on Cloud Computing (SoCC), pp.2:1–2:13, Oct. 2011.

[4] M. Koibuchi, H. Matsutani, H. Amano, D.F. Hsu, and H. Casanova, "A Case for Random Shortcut Topologies for HPC Interconnects," Proc. International Symposium on Computer Architecture (ISCA), pp.177–188, June 2012.
[5] A. Singla, C.Y. Hong, L. Popa, and P.B. Godfrey, "Jellyfish: Networking Data Centers Randomly," Proc. USENIX Symposium on Network Design and Implementation (NSDI), pp.225–238, April 2012.
[6] T. Skeie, O. Lysne, and I. Theiss, "Layered Shortest Path (LASH) Routing in Irregular System Area Networks," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2001.
[7] T. Skeie, O. Lysne, J. Flich, P. Lopez, A. Robles, and J. Duato, "LASH-TOR: A Generic Transition-Oriented Routing Algorithm," Proc. 10th International Conference on Parallel and Distributed Systems (ICPADS), pp.595–604, July 2004.
[8] R. Kawano, H. Nakahara, S. Tade, I. Fujiwara, H. Matsutani, M. Koibuchi, and H. Amano, "ACRO: Assignment of Channels in Reverse Order to Make Arbitrary Routing Deadlock-free," Proc. 15th IEEE/ACIS International Conference on Computer and Information Science (ICIS), pp.565–570, June 2016.
[9] J. Kim, W.J. Dally, and D. Abts, "Flattened Butterfly: a Cost-Efficient Topology for High-Radix Networks," Proc. International Symposium on Computer Architecture (ISCA), pp.126–137, June 2007.
[10] J. Kim, W.J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," Proc. International Symposium on Computer Architecture (ISCA), pp.77–88, June 2008.
[11] Scott Parker, "BG/Q Architecture," Mira Performance Boot Camp 2013, https://www.alcf.anl.gov/files/bgq-arch_0.pdf
[12] Y. Ajima, T. Shimizu, and S. Sumimoto, "Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers," IEEE Computer, vol.42, no.11, pp.36–40, Nov. 2009.
[13] M. Miller and J. Širáň, "Moore Graphs and Beyond: A survey of the Degree/Diameter Problem," Electronic Journal of Combinatorics, vol.20, no.2, pp.1–92, Nov. 2013.
[14] P.R. Hafner, "The Hoffman-Singleton Graph and its Automorphisms," Journal of Algebraic Combinatorics, vol.18, no.1, pp.7–12, 2003.
[15] W. Bao, B. Fu, M. Chen, and L. Zhang, "A High-Performance and Cost-Efficient Interconnection Network for High-Density Servers," Proc. IEEE 10th International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), pp.1246–1253, Nov. 2013.
[16] P.R. Hafner, "Geometric realisation of the graphs of McKay–Miller–Širáň," Journal of Combinatorial Theory, Series B, vol.90, no.2, pp.223–232, March 2004.
[17] M. Besta and T. Hoefler, "Slim Fly: A Cost Effective Low-Diameter Network Topology," Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp.348–359, Nov. 2014.
[18] "Graph golf: The order/degree problem competition," http://research.nii.ac.jp/graphgolf/
[19] U.Y. Ogras and R. Marculescu, ""It's a Small World After Al,": NoC Performance Optimization Via Long-Range Link Insertion," IEEE Transactions on Very Large Scale Integration Systems (TVLSI), vol.14, no.7, pp.693–706, July 2006.
[20] F. Silla and J. Duato, "High-Performance Routing in Networks of Workstations with Irregular Topology," IEEE Transactions on Parallel and Distributed Systems (TPDS), vol.11, no.7, pp.699–719, July 2000.
[21] J. Domke, T. Hoefler, and W.E. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp.616–627, 2011.
[22] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Transactions on Computers (TC), vol.C–36, no.5, pp.547–553, May 1987.
[23] D.M. Chiu, M. Kadansky, R. Perlman, J. Reynders, G. Steele, and M. Yuksel, "Deadlock-free Routing Based on Ordered Links," Proc. 27th Annual IEEE Conference on Local Computer Networks (LCN), pp.62–71, 2002.
[24] D.J. Watts and S.H. Strogatz, "Collective dynamics of 'small-world' networks," Nature, vol.393, no.6684, pp.440–442, June 1998.

Ryuta Kawano received the BE and ME degrees from Keio University, Yokohama, Japan, in 2013 and 2015, respectively. He is currently a PhD course student in the Department of Information and Computer Science, Keio university. His research interests include the area of high-performance computing and interconnection networks.

Hiroshi Nakahara received the BE degree from Keio University, Yokohama, Japan, in 2015. He is currently a master student in the Department of Information and Computer Science, Keio University.

Seiichi Tade received the ME degree from Keio University, Yokohama, Japan, in 2016.

Ikki Fujiwara received the BE and ME degrees from Tokyo Institute of Technology, Tokyo, Japan, in 2002 and 2004, respectively, and received the PhD degree from the Graduate University for Advanced Studies (SOKENDAI), Tokyo, Japan, in 2012. He is currently a Senior Researcher in Data-driven Intelligent System Research Center, Universal Communication Research Institute, National Institute of Information and Communications Technology, Kyoto Japan. His research interests include the areas of distributed systems and optimization. He is a member of the IEEE.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently an assistant professor in the Department of Information and Computer Science, Keio University. From 2009 to 2011, he was a research fellow in the Graduate School of Information Science and Technology, The University of Tokyo, and awarded a Research Fellowship of the Japan Society for the Promotion of Science.

Michihiro Koibuchi received the BE, ME, and PhD degrees from Keio University, Yokohama, Japan, in 2000, 2002 and 2003, respectively. Currently, he is an associate professor in the Information Systems Architecture Research Division, National Institute of Informatics and the Graduate University of Advanced Studies, Tokyo, Japan. His research interests include the area of high-performance computing and interconnection networks. He is a member of the IEEE and a senior member of IEICE and IPSJ.

Hideharu Amano received the PhD degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.