PAPER

Optimizing Slot Utilization and Network Topology for Communication Pattern on Circuit-Switched Parallel Computing Systems

Yao HU[†a)], ***Nonmember and*** **Michihiro KOIBUCHI**[†], ***Member***

SUMMARY In parallel computing systems, the interconnection network forms the critical infrastructure which enables robust and scalable communication between hundreds of thousands of nodes. The traditional packet-switched network tends to suffer from long communication time when network congestion occurs. In this context, we explore the use of circuit switching (CS) to replace packet switches with custom hardware that supports circuit-based switching efficiently with low latency. In our target CS network, a certain amount of bandwidth is guaranteed for each communication pair so that the network latency can be predictable when a limited number of node pairs exchange messages. The number of allocated time slots in every switch is a direct factor to affect the end-to-end latency, we thereby improve the slot utilization and develop a network topology generator to minimize the number of time slots optimized to target applications whose communication patterns are predictable. By a quantitative discrete-event simulation, we illustrate that the minimum necessary number of slots can be reduced to a small number in a generated topology by our design methodology while maintaining network cost 50% less than that in standard tori topologies.

key words: *parallel computing, interconnection network, circuit switching, time division multiplexing (TDM), end-to-end latency*

1. Introduction

Existing parallel computers are predominantly connected by packet-switched networks. The packet-based networks are mature and their bandwidth can increase by active optical cables, e.g., 100Gbps on InfiniBand EDR [1]. However, a conventional packet switch has 40–100ns delay and is difficult to further reduce its processing latency [2]. Current applications require a highly parallel processing that exchanges a large number of small data communications between processing cores. Such a way sometimes becomes a performance bottleneck in terms of latency rather than bandwidth due to congestion in traditional packet-switched networks. Therefore, there is a strong requirement for low-latency communication on parallel computing, and its demand continues to increase as the number of processing cores becomes large.

In a radical departure from conventional practice, we argue for the merits of circuit-switched (CS) networks in the design of future parallel computers. Unlike packet-switched networks that carry traffic flows in packets, circuit-switched networks establish connections that allocate network resources along a specified path from source to destination. After a connection is established, data is transferred without interruption until the connection is torn down. Our goal in this work is to optimize the circuit switching interconnection for parallel computers, which can simultaneously improve performance and efficiency while drastically reducing networking costs.

The prior works [3], [4] have shown that a circuit-switched network offers isolated predictable performance. First, data flows are transferred with guaranteed bandwidth, bounded latency and low jitter, and no data can be dropped due to congestion. Second, data flows do not interact with one another. The actions of one user's application can not be allowed to interfere with performance received by another user. Despite these benefits, a traditional circuit-switched network falls out of favor due to several perceived disadvantages relative to packet-switched networks: (1) High setup overhead. A connection must be established before data is transmitted. The time used for establishing the connection punishes the whole network latency performance. This is severe in a large-scale network. (2) Low link efficiency. Unused capacity guaranteed to a connection cannot be used by other connections on the same network. Thus, it has low link utilization due to its inability to perform statistical multiplexing.

For parallel computers, however, we argue that these perceived disadvantages of circuit-switched networking can be much restrained. In parallel computers, the interconnection networks are usually set up and maintained by one central organization that has full knowledge of topology and link capacities. For a circuit-switched network, this vastly simplifies the complexity and task of rerouting traffic during failures. Moreover, in parallel computers there are a bounded number of cores and nodes. Therefore, the communication paths are usually short and the round trip delay is low. In this study, we adopt a CS network with time division multiplexing (TDM) for improving link utilization while not consuming extra time to establish the connection.

This paper is based on our previous work [5], which has proposed a case for circuit switched (CS) networks realizing fast circuit switching with little overhead. Our target CS architecture provides the common benefits of a circuit-switched network, e.g., a dedicated communication path

Manuscript received June 26, 2018.
Manuscript revised October 4, 2018.
Manuscript publicized November 16, 2018.

[†]The authors are with the Information Systems Architecture Science Research Division, National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: huyao@nii.ac.jp

DOI: 10.1587/transinf.2018EDP7225

with a certain amount of bandwidth for each communication pair, hence the network congestion never happens and the end-to-end latency can be guaranteed or predicted. Furthermore, in our target CS network, multiple time slots in one switch can be exploited at the same time interval by different communications in cycles to maximize bandwidth utilization. On the other hand, to achieve optimum communication performance, the number of allocated time slots in one switch should be as small as possible because it is a direct factor to affect the end-to-end latency. To mitigate the frequent high setup overhead of a circuit-switched network relative to a packet-switched network, we perform statistical multiplexing of concurrent circuit connections which can be tunneled through just one circuit. To further mitigate the impact of the path setup overhead, we can use the same flow setup and tear down technique as that in [4], which is beyond the scope of this work.

Our recommendation for target applications is that their communication traffic patterns should be predictable and the number of source-destination communication pairs should not be large. This is because the bandwidth assigned to each source-destination communication pair increases as the maximum number of the communication pairs that go through a link becomes small. Reversely, if their traffic patterns are unknown, we have to prepare all-to-all communication circuit paths that are costly in terms of bandwidth.

Our work is not the first proposal to introduce circuit switching to parallel computing networks. For example, two recent proposals (Helios [6] and c-Through [7]) have employed hybrid electric/optical switch technologies to boost the performance of datacenter networks. The AN3 network by Microsoft [4] advocates the use of standalone circuit switching for datacenters in terms of advantages of capital cost and performance. Our approach is similar, but we put focus on the optimization of slot utilization and interconnection network for CS rather than simply applying a known hierarchical topology to have approximately equivalent bisection bandwidth and delay.

Typical parallel scientific applications perform well on k-ary n-cube topologies (e.g., numerical linear algebra kernels on 2-D or 3-D tori). By contrast, parallel applications that have irregular and/or dynamically evolving communication patterns require low average network latencies across all switch pairs [8]. These applications can perform poorly on k-ary n-cube topologies due to long shortest path lengths between some switches, but they are well-suited to random topologies [9]. It is thus reasonable to expect that future parallel computers will require different network topologies to support both legacy and emerging applications [10]. Given that different applications benefit from different topologies, in this work we optimize the target CS architecture and propose a dynamic network topology generator according to a given communication pattern and the maximum switch degree. This approach helps to incrementally reduce the minimum necessary number of slots in the network, and thus can reduce the end-to-end latency for each communication pair.

Our main contributions in this paper are as follows:

- We make static analysis and optimization of time slot utilization in our target CS network.
- We propose a CS topology generator to dynamically generate a specific interconnection network according to a given communication pattern and the maximum switch degree.
- By performing a quantitative discrete-event simulation, we present the advantages of the generated network topologies by our design methodology in terms of cost and efficiency.

The rest of this paper is organized as follows. Background information and related work are discussed in Sect. 2. Section 3 describes the static analysis of slot utilization and its optimization in our target CS network. Section 4 proposes an efficient CS network topology generator. Section 5 shows simulation results. Section 6 concludes with a summary of our findings in this paper.

2. Background and Related Work

2.1 Hybrid Packet-Optical Circuit Switch (OCS) Network

To overcome the limitations of traditional packet-based network architectures for huge traffic loads and variable traffic patterns, the hybrid packet-optical circuit switch (OCS) network [11]–[13] was first envisioned by researchers a decade ago and is now being realized in commercial datacenters. In the hybrid packet-OCS network, optical circuit switches are installed to augment packet-based switching to create a hybrid solution and offer the capability to handle large persistent data flows with high bandwidth. It also offers low latency (less than 60ns), which is very important to modern latency-sensitive applications.

The link setup time of an optical circuit switch (e.g., by using micro electro-mechanical systems (MEMS) [17] technologies) is typically 25ms [14], [16] (e.g., commercially available products [15]), which is required for electrostatic repositioning of micro-mirrors to achieve path switching. In the packet world, 25ms seems rather high. To reduce the large switching time from the aspect of software, recently several works [18], [19] have designed multiport microsecond control panels for optical circuit switching in datacenter networks. Besides, network management scripts or software-defined network (SDN) is used to redirect data flows from one network to another [16]. This would result in high network design cost for hardware and software management, since currently the OCS technology is not mature.

In this context, we attempt circuit switching to set up per-flow circuits, which are much more fine-grained than the circuits that are shared among many flows in hybrid switches. Surprisingly, we found out that the number of source-destination communication pairs is small in some parallel applications. The circuit switching can assign them to each time slot at the middle size of parallel computers. This provides ideal low-latency and guaranteed bandwidth communication only by circuit switching. Reversely, we do

not have to employ an electric packet network in addition to circuit switching under such an environment. Consequently, we challenge to make an interconnection network only using circuit switching for parallel computers in this paper.

2.2 Arrayed Waveguide Grating Router (AWGR)

Arrayed waveguide grating router (AWGR) [20]–[23] has been researched for *all-to-all* connections [24], in which every processor sends a unique message to any other processor at any time. *All-to-all* is the densest communication pattern that can be imposed on a computing network. AWGR uses wavelength division multiplexing (WDM) technology for frequency domain parallelism and allows for the multiplexed wavelengths in the waveguides which are separated and cross-connected.

For an AWGR, m nodes respectively connected to m input ports can use m wavelengths to reach different output ports simultaneously without interfering with each other. However, despite the intrinsic merit of dense interconnection, its port count is usually restricted by size, fabrication constraints and inter-channel crosstalk. Therefore, the need arises for an *all-to-all* interconnection architecture using AWGRs with reduced or limited number of wavelengths.

In our work, the need of reducing the number of time slots for every switch seems similar to that of reducing the number of wavelengths for every AWGR. However, they differ in two main aspects. First, AWGR assumes non-blocking *all-to-all* switching, while our work applies to various communication patterns in the network. Second, AWGR reduces the number of wavelengths by deploying one or more AWGRs, while in our work the proposed network topology generator reduces the number of time slots by decentralizing traffic over the whole network.

2.3 Non-Blocking Network

Contention in the network increases latency and decreases bandwidth substantially, especially for long messages [25]. Some applications have large payload sizes for point-to-point communications and are highly susceptible to network contention. If not well controlled, contention could have a large negative impact on the performance of parallel programs even if it occurs only within a small portion of the network. This is because a stalled process may slow down others that are communicating with it in lock-step fashion. It has been shown that this could account for as much as 30% degradation in performance [25].

Conflict-free or non-blocking networks have been extensively researched for general-purpose processing and specific communication patterns like *multicast* and *all-to-all broadcast*. For example, a way of supporting all permutations by dividing a message into smaller fragments and distributing them according to a routing matrix in two passes through the network is described in [26]. However, one problem with this approach and others like it is scalability, since the number of fragments grows with the size of the network and extra passes are required. This may employ excessive resources.

In this work, we propose a design methodology for finding a specific network topology according to a given communication pattern and the maximum switch degree. The methodology addresses the problem of optimizing spatial sharing of communication resources and spatial overlap of messages. As the first step, we construct a network topology prototype based on a recursive bisection technique via systematic partitioning. This method is similar to some previous on-chip/off-chip interconnection works [27], [28]. For example, minimizing the number of time slots is like pursuing contention-free communication. However, we do not target totally conflict-free networks where the number of time slots for each switch is only one. Also, we do not solve the coloring problem [28] when counting the number of necessary communications between two network partitions, but measure how to balance network traffic to reduce the minimum necessary number of slots in the network. Besides, in the study [27] it is required to check the design constraint (i.e., the maximum switch degree) after each node move, while in our work such procedure can be avoided and the design constraint is constantly met, because the degree of every switch keeps the same after each node move.

3. Slot Allocation on Circuit Switching

3.1 Target Circuit Switch

The target circuit switch [5] is depicted in Fig. 1. One port of the switch is connected to a compute node or a neighboring switch. Each port consists of one or several links and each link consists of several time slots (buffers). The volume of one time slot (buffer) is the size of the data received and accommodated within one cycle. The connection between an input slot and an output slot is established before the data transfer. Updating of the connection is supported after circuit reconfiguration. The read or write operation on the input or output side tours the time slots one by one in cycles. On the same time slot, the read and write operations are synchronized with the same frequency, thus no conflict like write-after-read or read-after-write occurs.

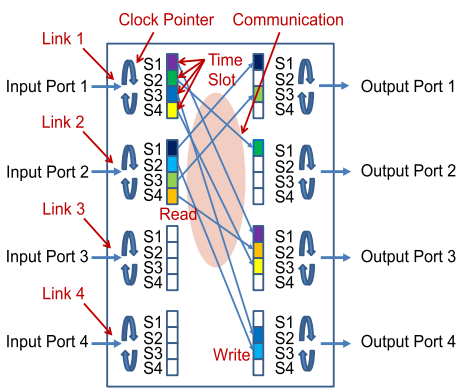


Fig. 1 Switch design for target circuit-switched (CS) networks.

If there are n time slots allocated to one link, and a communication occupies m ($m < n$) slots, its bandwidth is m/n link bandwidth. Therefore, the end-to-end latency of any communication in the network can be guaranteed or predictable. The most important thing for the switch design is to get the minimum necessary number of slots installed in one switch. On one hand, a large number of time slots can cause large end-to-end latency because of iterative time slot access. On the other hand, a small number of time slots may result in long queuing or even conflict among multiple communications at the same switch port. When a CS network is composed of identical switches installing the same number of time slots, the minimum necessary number of slots in any switch is a direct factor to affect the whole network latency performance. In this work, a main target for the CS network design is to reduce the minimum necessary number of slots and thus to decrease the communication latency.

We use the discrete-event simulation framework SimGrid (v3.12) [32] to evaluate the benchmark performance on conventional packet switched (PS) networks and our target circuit switched (CS) networks. SimGrid implements validated simulation models, is scalable, and makes it possible to simulate the execution of unmodified parallel applications that use the message passing interface (MPI).

We assume two interconnection networks for conventional all-packet transmission, i.e., 3-D torus and fully-connected. For both the interconnection networks, the link bandwidth is set to 400Gbps/1TGbps and the switch delay is set to 200ns. In fully-connected CS networks, one slot is assumed to occupy 1× bandwidth of 25Gbps and the switch delay is assumed to be 10ns in an ideal case that no communication conflict occurs at any time slot so that the link bandwidth can be maximumly utilized by all communications. The computation power of each node is set to 1TFlops.

Figure 2 presents evaluation results of execution time comparison between conventional 3-D torus/fully-connected PS networks and target fully-connected CS networks. We show reciprocals of actual execution times, thus higher values are better. We make the case of 400Gbps 3-D torus PS as baseline. Because the average hop count of communications over 3-D torus is larger than that over fully-connected networks, it performs the worst even with the link bandwidth of 1TGbps. The fully-connected CS significantly outperforms 3-D torus PS and even performs advantage over fully-connected PS due to its fine-grained circuit switching mechanism. For different benchmarks, the CS networks perform better when allocated more slots for one switch, because the link bandwidth is increased for communications while only a bit of switch delay is imposed.

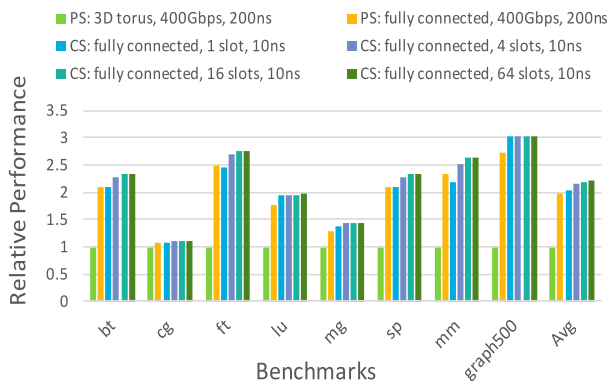


Fig. 2 Relative performance of execution times. Values are reciprocal (higher is better).

In our previous study [5], we showed a case of circuit switched network for parallel computers and made a static analysis of various communication patterns over 2-D mesh. We extend the previous work mainly from three aspects in this study using the target circuit switch. First, we improve slot utilization by the proposed optimization methods, and evaluate the efficiency over mesh, torus and fat-tree interconnection networks. Second, we investigate the tradeoff between the number of time slots and the number of network resources when designing our circuit switched topology generator, and make a comparison with mesh and torus in terms of resource utilization efficiency. Third, we evaluate the performance of topology reconfiguration in our target circuit switched network.

3.2 Static Analysis of Communication Patterns

We analyze the minimum necessary number of slots (N) of all switches with various famous communication patterns over the CS network.

The value of N is equal to the maximum number of overlapped communications on the same link, if one communication consumes one slot. In this work, we calculate the maximum number of overlapped communications on the same link by using a modified version of an on-chip data transfer algorithm [29]. Firstly, we divide a parallel application into a set of small parallel tasks, which are mapped to respective node pairs as the communication pattern. Secondly, for all communication node pairs we calculate the communication path from a source node to its destination node. Finally, we get the maximum number of overlapped communications going through the same switch port, which is equal to the value of N for the switch and network design.

Figure 3 shows a 2-D mesh (4 × 4) network topology

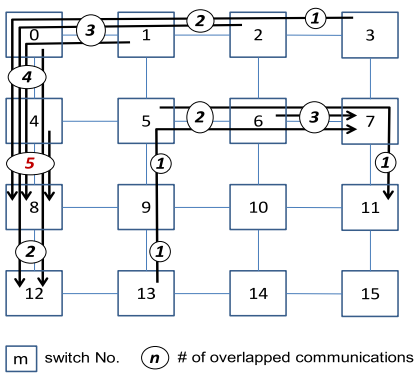


Fig. 3 The minimum necessary number of slots (N) with an example communication pattern (part) in a 2-D mesh network.

Table 1 The minimum necessary number of slots (N) for different communication patterns.

# of end nodes	uniform	bit reversal	matrix transpose	neighbor	perfect shuffle	butterfly	bit complement	tornado
16 (2-D mesh, 4×4)	4	3	3	3	2	2	2	2
64 (2-D mesh, 8×8)	6	7	7	4	4	4	4	4
256 (2-D mesh, 16×16)	11	15	15	3	8	8	8	8
1024 (2-D mesh, 32×32)	14	31	31	4	16	16	16	16
4096 (2-D mesh, 64×64)	28	63	63	7	32	32	32	32
4096 (3-D mesh, $16 \times 16 \times 16$)	13	15	48	8	8	8	8	8
4096 (4-D mesh, $8 \times 8 \times 8 \times 8$)	9	56	56	5	4	4	4	4

Table 2 The minimum necessary number of slots (N) for *all-to-all* communications in 2-D mesh networks.

Routing	16-node	64-node	256-node
destination-based	16	128	1024
path-based (all scatter)	36	528	8256
path-based (all broadcast)	8	32	128
tree-based (all scatter)	12	56	240
tree-based (all broadcast)	12	56	240

with an example communication pattern. Let $c(s, d)$ denote the communication from switch s to switch d, and let $l(u, v)$ denote the link from switch u to switch v. Note that we assume *full-duplex* links, that is, two communications going along opposite directions do not interfere with each other and they are treated separately. Therefore, the communication $c(s, d)$ is not equal to the communication $c(d, s)$ and the link $l(u, v)$ is not equal to the link $l(v, u)$. In this network, the maximum number of overlapped communications $c(0, 12)$, $c(1, 8)$, $c(2, 12)$, $c(3, 8)$ and $c(4, 8)$ on the same link $l(4, 8)$ is 5, thus $N = 5$.

Table 1 shows the values of N in different networks with various communication patterns [30], [31] including *uniform*, *bit reversal*, *matrix transpose*, *neighbor*, *perfect shuffle*, *butterfly*, *bit complement* and *tornado*. We assume that one end node is connected to one switch. It can be seen that, as the network size increases, the value of N becomes large. For example, for the communication pattern of *uniform*, at least 4, 6, 11, 14 and 28 time slots are required if a 2-D mesh network is composed of 16, 64, 256, 1024 and 4096 switches, respectively. If the network size is the same, a higher-dimension network requires less number of slots. For instance, in 4096-node networks with the *uniform* communication pattern, a 2-D mesh topology, a 3-D mesh topology and a 4-D mesh topology require at least 28, 13 and 9 slots, respectively.

To test full potential network communication performance, we also make similar static analysis for *all-to-all* communications in 2-D mesh networks, as shown in Table 2. We use the following different routing methods to support *all-to-all* communications.

- *Destination-based*: routing traffic only based on the destination
- *Path-based*: routing traffic along a previously specified path towards the destination
- *Tree-based*: routing traffic along a tree-like path towards the destination, where the source is the root of the tree

All scatter refers to the case that a source sends different messages to different destinations individually, while *all broadcast* refers to the case that a source sends the same message to all destinations at once.

From the analysis results, it can be seen that the value of N varies according to the used routing method in the network. Therefore, the whole network has a great potential to be optimized by a proper way so that the value of N can be reduced to a small number.

3.3 Optimization of Slot Utilization

So far we have assumed that one communication uses only one time slot for data transfer. As shown in Fig. 3, the five communications $c(0, 12)$, $c(1, 8)$, $c(2, 12)$, $c(3, 8)$ and $c(4, 8)$ go through the same link $l(4, 8)$, thus they have no improvement room to be allocated more slots if not increasing the value of N. However, for example, if any communication of $c(5, 11)$, $c(6, 7)$ and $c(13, 7)$ is allocated one more slot, the number of required time slots on the local congested link $l(6, 7)$ becomes $3 + 1 = 4$, which is still smaller than the value of N (5). In this case, the slot utilization is improved but not increasing the value of N over the network.

In order to improve slot utilization, we try to allocate more than one slot to one communication while not increasing the value of N in the network. We take two steps. The first step is that we decide which communication is picked to be allocated more slots in the network. We consider the following three options to give priorities to selection among communications that can be improved.

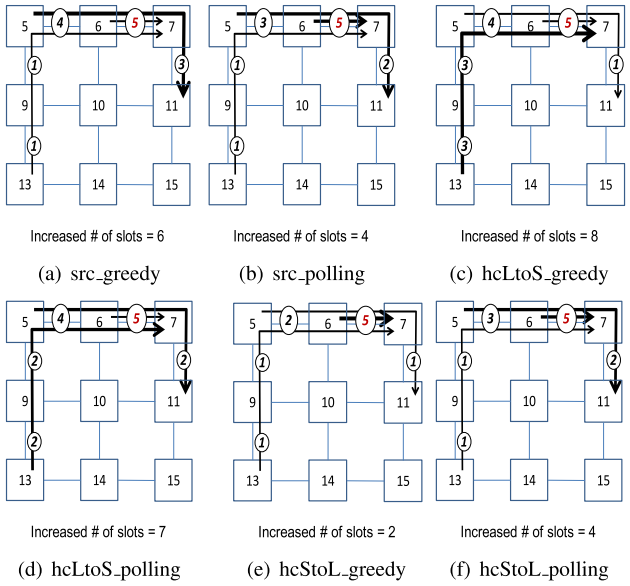
- *src*: in natural order of source node number
- *hcLtoS*: in descending order of hop count
- *hcStoL*: in ascending order of hop count

The second step is that for each communication we consider how many slots are incrementally allocated. We consider the following two ways with tradeoff between efficiency and fairness for all communications.

- *greedy*: current communication is allocated as more slots as possible
- *polling*: current communication is allocated one more slot at a time

Therefore, we can compare six methods for improving slot utilization in the network: *src_greedy*, *src_polling*, *hcLtoS_greedy*, *hcLtoS_polling*, *hcStoL_greedy* and *hcStoL_polling*.

Figure 4 shows the six optimization methods within a 9-switch subnetwork in the previous example. For each



(a) src_greedy (b) src_polling (c) hcLtoS_greedy

(d) hcLtoS_polling (e) hcStoL_greedy (f) hcStoL_polling

Fig. 4 Optimization of slot utilization ($N = 5$). A bold arrow represents a communication that is allocated multiple slots. A number in a circle represents # of slots used by the link. Unrelated switches and their connections are omitted.

switch, because the input and output ports are symmetric, we only count the number of slots on the output side. Since $N = 5$ over the network, the communications $c(5, 11)$, $c(6, 7)$ and $c(13, 7)$ have 2-slot improvement room. The communications $c(0, 12)$, $c(1, 8)$, $c(2, 12)$, $c(3, 8)$ and $c(4, 8)$ are omitted because they have no improvement room. The hop count of a communication is defined as the number of switches that the communication goes through. From the analysis results, it can be seen that the increased number of used slots after optimization varies according to the optimization method. An exception is that the result for *src_polling* is the same as that for *hcStoL_polling* because they both happen to first pick $c(5, 11)$ and then pick $c(6, 7)$ to allocate one more slot.

4. Network Topology Generator

Even though we have applied above optimization methods to increase the number of used slots, the slot utilization is still low because the communication traffic load is not balanced and causes resource waste and inefficiency over the network. To decentralize local heavy communication traffic and further reduce the value of N, we propose a method to automatically generate a network topology according to a given communication pattern and the maximum switch degree.

Our method is similar to a previous on-chip interconnection work [27], which addresses the problem of optimizing spatial sharing of communication resources and spatial overlap of messages. Our proposed network topology generator uses a recursive bisection technique to systematically determine what topology with the minimal set of resources is used to efficiently support a target communication pattern.

Algorithm 1 Generates network topology by *comm. pattern* and *max. switch degree*.

```
1: procedure TOPOGEN(CommPattern cp, MaxSwDegree sd)
2:     Initiate topo with a single switch connecting all nodes
3:     topo = partitionSwitches(sd, topo)   // → Sect. 4.1
4:     topo = swapNodes(cp, topo)   // → Sect. 4.2
5:     topo = optimizeLinks(cp, topo)   // → Sect. 4.3
6:     return topo
7: end procedure
```

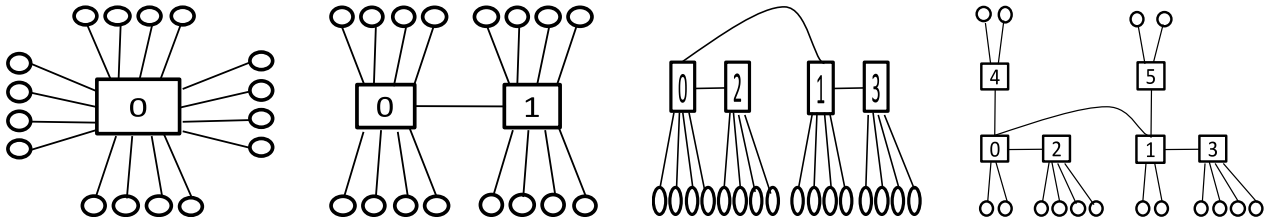
The design methodology makes use of static analysis of the target communication pattern to reduce link conflicts among communications while using as few network resources (e.g., switches and links) as possible. Generally, a number of different design constraints can be used. A simple one used in this work is the maximum switch degree, which limits the number of ports of each switch to be less than or equal to a specified value. The design constraint on switch degree is due to the limited number of switch ports. Although minimizing the number of time slots is like pursuing contention-free communication in the work [27], we do not target totally conflict-free networks where the number of time slots for each switch is only one.

Our topology generation algorithm begins with a single *giant* switch connecting all end nodes like a crossbar. Obviously, the degree of the *giant* switch is equal to the total number of the end nodes. The essential idea of the design methodology is to systematically partition the *giant* switch into smaller ones which all meet the design constraint on switch degree. This is done by recursive bisection (Sect. 4.1). Afterwards, we try to swap the placement of the end nodes attached to different switches to optimize network partitions (Sect. 4.2). We also try to determine the best routes for communications to meet the requirement of a target communication pattern, e.g., by diverting part of conflict communications on the same link to different links, so that the value of N in the network can be reduced (Sects. 4.3 and 4.5).

Algorithm 1 shows the procedures of generating a custom network topology according to a given communication pattern and the maximum switch degree. For easy understanding, we explain the topology generation procedures by the following example assuming that the number of end nodes is 16 and the maximum switch degree is 5 in the network.

4.1 Switch Partition

Figure 5 presents a step-by-step illustration of switch partition processes. As described previously, the initial network constructed by a single *giant* switch connecting all end nodes is partitioned recursively until the specified design constraint, i.e., the maximum switch degree, is met by all switches. The first step is to connect all 16 nodes to the single switch 0, as shown in Fig. 5 (a). Obviously, switch 0 violates the design constraint, i.e., the switch degree is larger than 5. The second step is to partition switch 0 into



(a) Step 1 (Initiation) (b) Step 2 (Partition) (c) Step 3 (Partition) (d) Step 4 (Final prototype)

Fig. 5 Recursive switch partition (# of nodes = 16, maximum switch degree = 5). A rectangle with a number represents a switch with its switch number. A circle represents an end node.

Algorithm 2 *Partition switches*.

```
1: procedure PARTITIONSWITCHES(MaxSwDegree sd, Topology
   topo)
2:     while degree(topo) > sd do
3:         switches = getSwitches(topo)
4:         for sw in switches do
5:             if degree(sw) > sd and nodes(sw) > 1 then
6:                 sw1, sw2 = partition(sw)
7:                 allocateNodes(sw, sw1, sw2)
8:                 createLink(sw1, sw2)
9:                 for sw_n in neighbors(sw1) do
10:                    if sw_n != sw2 and degree(sw_n) < sd then
11:                        createLink(sw_n, sw2)
12:                    end if
13:                end for
14:            end if
15:        end for
16:        topo = update(topo)
17:    end while
18:    return topo
19: end procedure
```

two new switches 0 and 1, and to automatically assign half of nodes originally connected by switch 0 to connect switch 1. Switch 0 and switch 1 are then connected with each other, as shown in Fig. 5 (b). Still, both switch 0 and switch 1 violate the design constraint, as depicted in Fig. 5 (c), thus they are partitioned again like in the previous step. Because the degree of switch 1 is larger than 5, switch 2 is not connected to switch 1; likewise, switch 3 is not connected to switch 0. Similarly, half of nodes previously connected by switch 0 are connected with switch 2, and half of nodes previously connected by switch 1 are connected with switch 3. The partition algorithm terminates when all switches satisfy the design constraint, i.e., the degree of any switch in the network is less than or equal to 5. Figure 5 (d) presents the final topology prototype after switch partition, where the degree of each switch is not larger than 5. Since the degrees of the neighboring switches (except switch 4) of switch 0 are not less than 5, switch 4 is only connected to switch 0; likewise, switch 5 is only connected to switch 1. Algorithm 2 describes the switch partition processes according to the specified maximum switch degree in the network. It is worth noting that the switch partition algorithm is independent of the specified communication pattern.

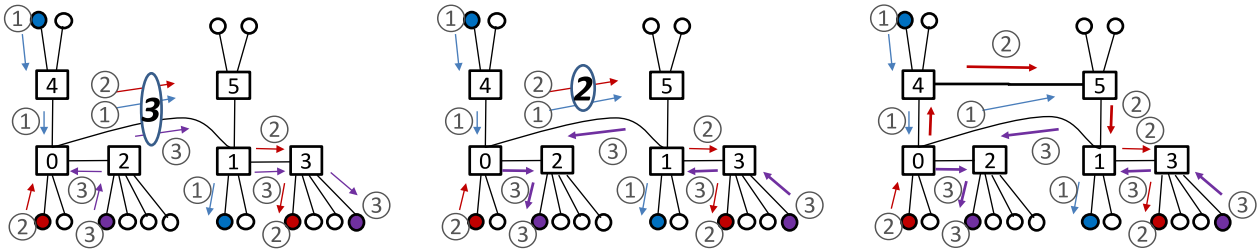
4.2 Node Swap

The next procedure is to optimize the network topology according to a given communication pattern. We consider to swap end nodes that cause the most crowded or congested link(s) in the network to reduce the value of N. Figure 6 (a) depicts the topology formed by previous network partition with an example communication pattern. In this network, $N = 3$, which is consumed by link $l(0, 1)$. Three communications ①, ② and ③ pass through $l(0, 1)$, thus we try to swap end nodes in one of the three communications. For instance, we swap end nodes in communication ③, as shown in Fig. 6 (b), so that the communication direction becomes opposite to the original one. Remember that two communications going along opposite directions do not interfere with each other and they are treated separately, thus the value of N is reduced to 2 in the network. If we swap end nodes in any of other two communications ① and ②, it turns out to be the same result, i.e., $N = 2$, we thus omit the illustration. Algorithm 3 shows the procedures of swapping end nodes that cause the most crowded link(s) in the network. Because the number of end nodes connecting a switch keeps the same after node swap, the whole network still meets the design constraint, i.e., the maximum switch degree does not change.

One thing to be noted is the case that there are more than one most crowded link in the network. In this case, we try to find one or more communications that pass through the union set of all these most crowded links and then swap the corresponding end nodes. If such a communication does not exist, we try to find communications that pass through as a large subset of these most crowded links as possible. Another thing to be noted is that the end nodes are swapped only when the value of N becomes smaller after node swap. If the value of N becomes larger, the corresponding end nodes are not swapped; if the value of N does not change, it is pending for the next procedure of link optimization (Sect. 4.3).

4.3 Link Optimization

We already know that after switch partition (Sect. 4.1) and node swap (Sect. 4.2), the degree of each switch in the network is not larger than 5. For the switches of which the



(a) After switch partition ($N = 3$). (b) After node swap ($N = 2$) (c) After link optimization ($N = 1$)

Fig. 6 An example of topology optimization for reducing the value of N in the network.

Algorithm 3 *Swaps nodes* that cause the most crowded link.

```
1: procedure swapNodes(CommPattern cp, Topology topo)
2:     nodes = findNodesByCrowdedLinks(cp, topo)
3:     for pair in nodes do
4:         topoNew = swapNodes(topo, pair)
5:         if getN(cp, topoNew) > getN(cp, topo) then
6:             continue
7:         else if getN(cp, topoNew) ==  getN(cp, topo) then
8:             pend(topoNew)
9:         else if getN(cp, topoNew) < getN(cp, topo) then
10:            topo = topoNew
11:        end if
12:    end for
13:    return topo
14: end procedure
```

Algorithm 4 *Optimizes links* and routes.

```
1: procedure optimizeLinks(CommPattern cp, Topology topo)
2:     sws = getSwitchesLessDegree(topo)
3:     for pair in sws do
4:         topoNew, linkNew = addLinks(topo, pair)
5:         routes = calculateRoutesByDijkstra(cp, topoNew)
6:         routesNew = routeViaNewLinks(cp, topoNew, linkNew)
7:         if getN(topoNew, routesNew) < getN(topoNew, routes)
   then
8:             setRoutes(topoNew, routesNew)
9:             topo = topoNew
10:        else
11:            setRoutes(topoNew, routes)
12:            topo = eraseLink(topoNew, linkNew)
13:        end if
14:    end for
15:    return topo
16: end procedure
```

degrees are smaller than 5, they may still have possibilities to help further reduce the value of N. Figure 6 (c) shows such a case. The degrees of switches 0, 1, 2 and 3 are 5, while the degrees of switches 4 and 5 are 3. Then we consider to create a connection between switch 4 and switch 5, and to divert the most crowded traffic in the network to this new connection. For instance, if communication ② diverts its traffic from the direct path (switch $0 \rightarrow 1$) to an indirect path (switch $0 \rightarrow 4 \rightarrow 5 \rightarrow 1$), there is no overlapped communication on any link in the network, i.e., $N = 1$.

In this work, we consider the following two cases for link optimization to reduce the value of N in the network.

- *indirect path*: communication hop count is increased after diverting traffic to a different path
- *parallel path*: communication hop count does not change after diverting traffic to a different path

Note that, after links added, we use the *Dijkstra* algorithm to recalculate the shortest path for every node pair, thus a *shortcut path* does not exist for link optimization. If the value of N is reduced after diverting traffic from the most crowded path to an *indirect path* or *parallel path*, we take it as a new route for the corresponding communication. Otherwise, we still use the old path and erase the new connection.

As link optimization goes on, communication traffic spreads over the whole network, and the maximum number of communications passing through the same link can decrease. Note that, we search for the most crowded or congested link(s) over the whole network, thus it is not a local optimal solution to reduce the value of N. In other words, we make global optimization by identifying the largest number of occupied time slots on a switch port within the same time interval in the network. The whole procedures of link optimization are described in Algorithm 4.

4.4 Time Complexity

The initial switch partition algorithm runs in $O(\log V)$ time, where V is the number of end nodes in the network. The procedure of node swap only considers a constant or limited number of moves between the partitions that cause the most crowded or congested link(s) in the network. After that, links and routes are optimized by using the Dijkstra algorithm to recalculate the shortest path for each node pair when links are added, which takes $O(E+V \log V)$ time when using Fibonacci Heap, where E is the number of links in the network. At the finalization of the topology generation, we find out the exact number of N and network resource cost such as the number of switches and links. Therefore, the overall complexity of the algorithm, dominated by the optimization of links and routes, is $O(E + V \log V)$.

The evaluation on the running time of the topology generator script for different network sizes written in Python 2.7 in a machine with Intel i7-6500U (2.50GHz) CPU and 16GiB Memory is shown in Table 3 (the evaluation conditions will be shown in Sect. 5.2).

Table 3 Running time of the topology generator script with different network sizes.

16 nodes	64 nodes	256 nodes	1024 nodes
0.012s	0.137s	0.229s	0.857s

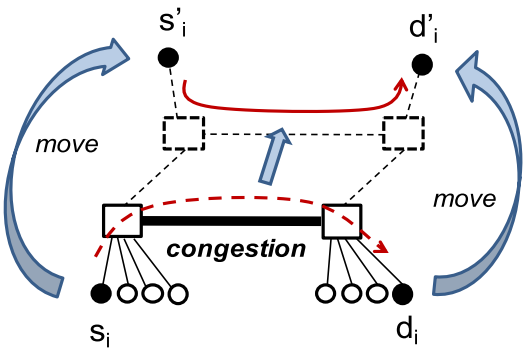


Fig. 7 The value of N can be reduced by increasing network resources and taking a bypass route for a communication pair.

4.5 Tradeoff Between N and Network Cost

So far we have developed a minimal topology according to a given communication pattern and the maximum switch degree, where the amount of network resources including switches and links can reach the smallest in our approach and the value of N can be also reduced. In this section, we discuss another possibility to reduce the value of N by increasing a certain number of network resources.

Figure 7 illustrates such an example of diverting a communication from a congested link to a newly added bypass route. This helps to further reduce the value of N in the network with a tradeoff between the value of N and network resource amount. For instance, end nodes s_i and d_i of communication $c(s_i, d_i)$ are moved to attach newly added switches (dashed boxes) to form a new communication $c(s'_i, d'_i)$. End nodes s'_i and d'_i and switches are connected by newly added links (dashed lines). Note that, in this way the added links can continue to bear more communications on other congested links (although not shown in Fig. 7), only if the added switches observe the network design constraint, i.e., the maximum switch degree. The degrees of the switches originally connected to nodes s_i and d_i can keep the same, because they respectively "lose" an end node while connecting to a new switch.

After the above procedures, the network traffic becomes more balanced and the value of N can be further reduced to a smaller number that we specify. Therefore, besides a given communication pattern and the maximum switch degree, we can also provide a value of N as an input parameter into our topology generator. The procedures are described in Algorithm 5.

Algorithm 5 Generates network topology based on *minimum necessary number of slots (N)*.

```
1: procedure topoGen_Rev(CommPattern cp, SwitchDegree sd, N n)
2:     topo = topoGen(cp, sd)
3:     sw1, sw2 = addSWs()
4:     while getN(topo) > n do
5:         if connected(topo, sw1, sw2) == false then
6:             topo = createLinks(topo, sw1, sw2)
7:         end if
8:         nodes = findNodesByCrowdedLinks(cp, topo)
9:         for pair in nodes do
10:            topo = moveNodes(pair, sw1, sw2)
11:            if degree(sw1, sw2) == sd then
12:                sw1, sw2 = addSWs()
13:                topo = createLinks(topo, sw1, sw2)
14:            end if
15:        end for
16:    end while
17:    return topo
18: end procedure
```

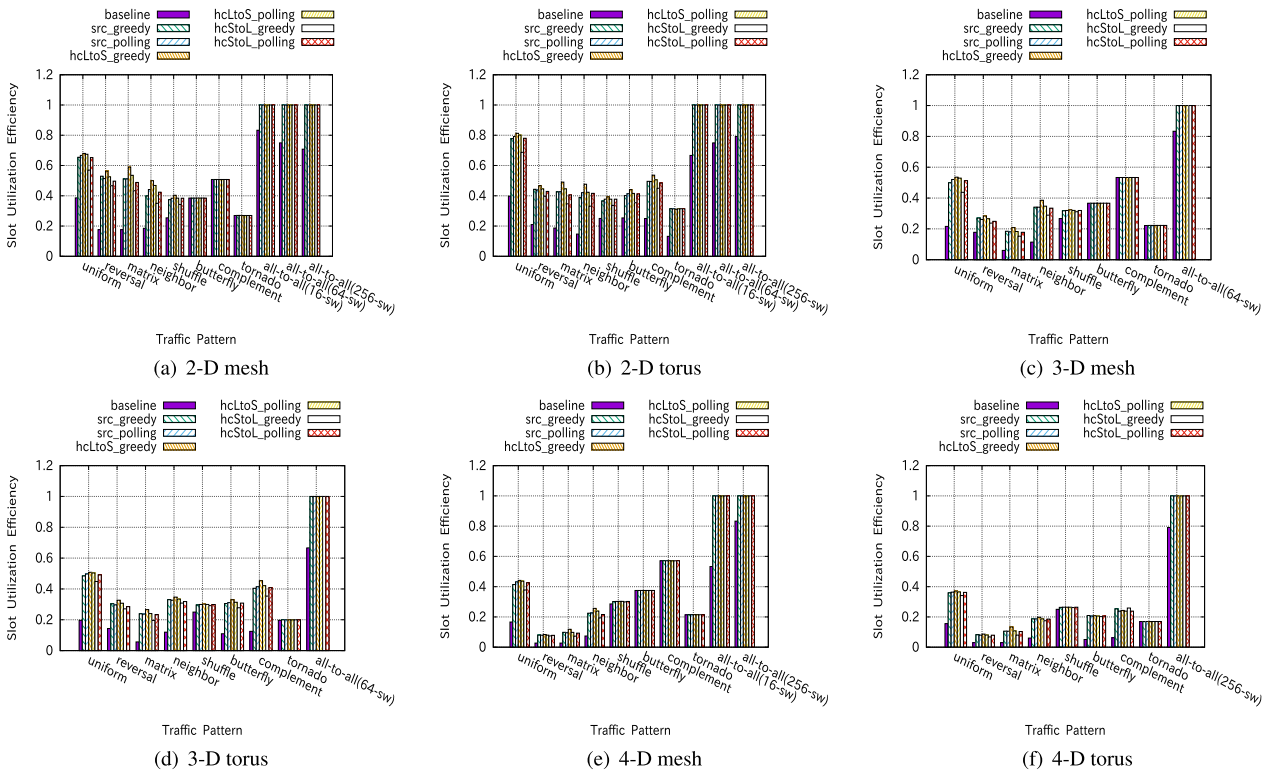
5. Evaluation

5.1 Slot Utilization

We take the case that one slot is allocated to one communication as *baseline*, and evaluate the slot utilization efficiency for the optimization methods as mentioned in Sect. 3.3. For each optimization method, the whole network will reach its full potential in terms of slot utilization when any communication is allocated one more slot the value of N will increase accordingly. The slot utilization efficiency is defined as the ratio of the number of used slots to the number of all switch slots in the network.

Figure 8 presents the slot utilization efficiency in mesh/torus networks. Overall, *hcLtoS_greedy* performs the best and *hcStoL_greedy* performs the worst among the six optimization methods, because the increased number of used slots for a communication at a time heavily depends on the hop count of the communication. It can be also seen that, for any communication pattern except *all-to-all*, the slot utilization efficiency is far below 100% even after optimization, thus it still has large improvement room to make the maximum use of switch slots in the network. The slot utilization efficiency for *all-to-all* can reach 100% with any optimization method, because any two nodes (switches) have communications and thus can be used to saturate the link utilization by occupying more time slots if required.

Except mesh and torus, we also evaluate the slot utilization efficiency in fat-tree networks as shown in Fig. 9. Figure 10 presents the evaluation results. As seen in Fig. 10 (a), compared to mesh/torus networks, the slot utilization efficiency is extremely low for all the communication patterns except for *neighbor* and *all-to-all*. The reason can be known from Fig. 10 (b), where the slot utilization efficiency of the *root* (first-layer) switch approaches or already reaches



(a) 2-D mesh (b) 2-D torus (c) 3-D mesh

(d) 3-D torus (e) 4-D mesh (f) 4-D torus

Fig. 8 Slot utilization efficiency in 4096-node mesh/torus interconnection networks.

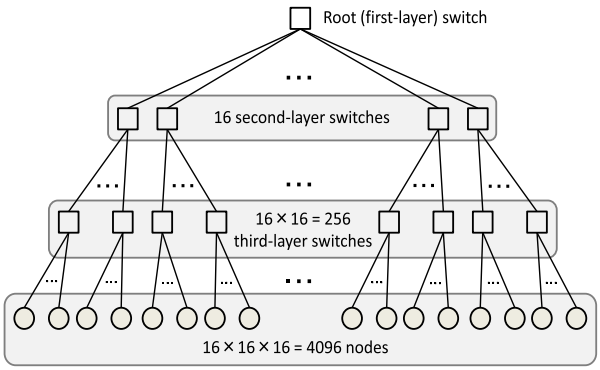


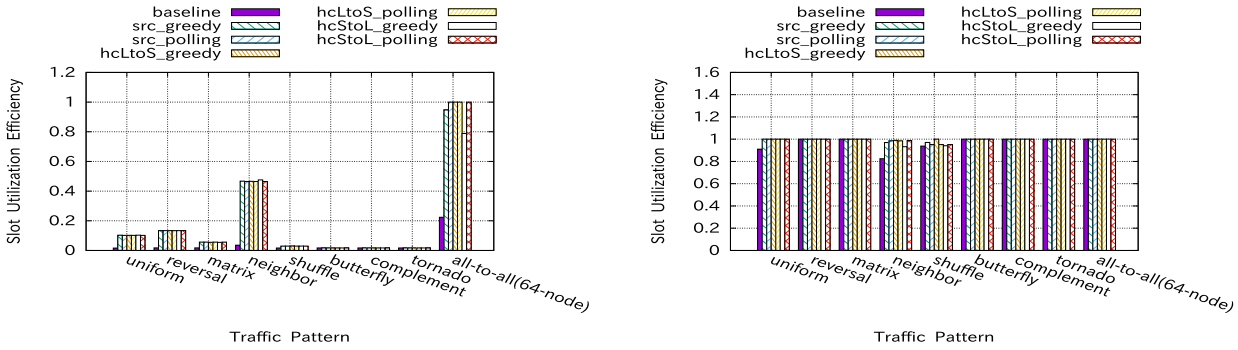
Fig. 9 Fat-tree topology with 4096 nodes.

100%. This means that most of the traffic for any communication pattern goes through the *root* switch, and if one more slot is allocated to any communication it probably increases the value of N. Therefore there is no improvement room for the overall slot utilization efficiency when not increasing the value of N. For this reason, the slot utilization efficiency can not reach 100% for *all-to-all* by *src_greedy* and *hcStoL_greedy*. Also, as shown in Fig. 10 (c) and Fig. 10 (d), except *all-to-all*, the slot utilization efficiency of the second-layer switches is low and that of the third-layer switches is even lower. For *neighbor*, since one node only communicates with its neighboring node, most of the traffic does not reach the *root* switch and it can be optimized much further than other communication patterns except for *all-to-all*. Therefore, the slot utilization efficiency for *neighbor* is obviously larger than that for other communication patterns except for *all-to-all*.

5.2 Network Cost Efficiency

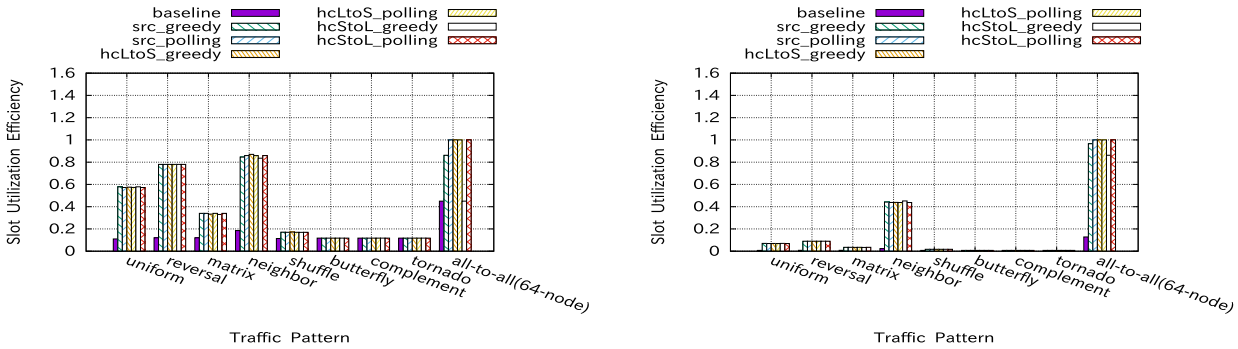
In this section, we evaluate the usefulness and effectiveness of the proposed CS topology design methodology using various well-behaved communication patterns, which are fed into our network topology generator written in Python 2.7 in a machine with Intel i7-6500U (2.50GHz) CPU and 16GiB Memory. Figure 11 depicts an example of the inter-switch topologies (nodes omitted) generated by our design methodology with different network sizes. In all cases of this example, the communication patterns are all assumed to be *uniform* and the maximum switch degrees are all set to 5. The generated topologies are minimal in terms of resource amount because we do not specify a value of N. As described in Sect. 4.5, we can further reduce the value of N for each generated topology by increasing a certain number of switches and links. The larger amount of resources we deploy, the smaller value of N we can obtain in the network.

In our evaluation, the performance of generated topologies is compared with that of other traditional standard networks such as tori in terms of resource usage. We assume the same maximum switch degree in our generated topologies as that in standard tori networks, so that we can make straightforward comparison between them. The evaluation results are shown in Fig. 12. Overall, as shown from Fig. 12 (a) to Fig. 12 (c), the generated topologies outperform the counterpart 2-D tori topologies in terms of resource



(a) All switches

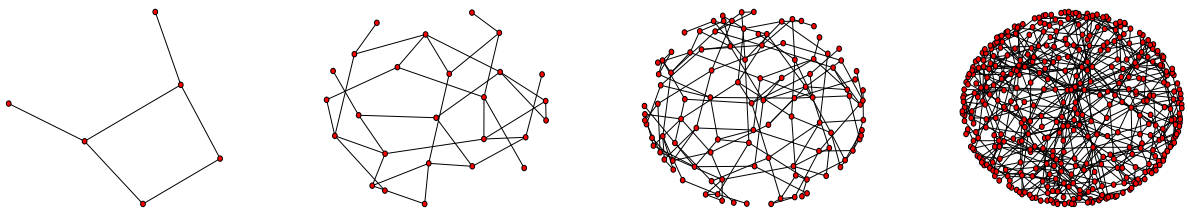
(b) The root (first-layer) switch



(c) The second-layer switches

(d) The third-layer switches

Fig. 10 Slot utilization efficiency in 4096-node fat-tree interconnection networks.



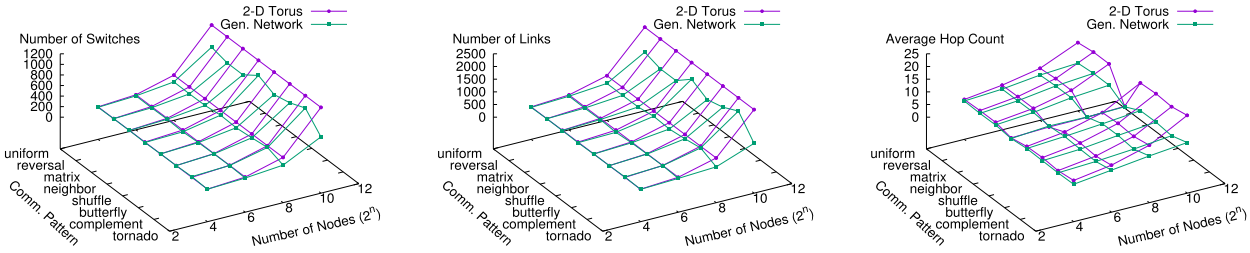
(a) 16 nodes (b) 64 nodes (c) 256 nodes (d) 1024 nodes

Fig. 11 The inter-switch topologies generated with different network sizes (communication pattern = *uniform*, maximum switch degree = 5). End nodes are omitted.

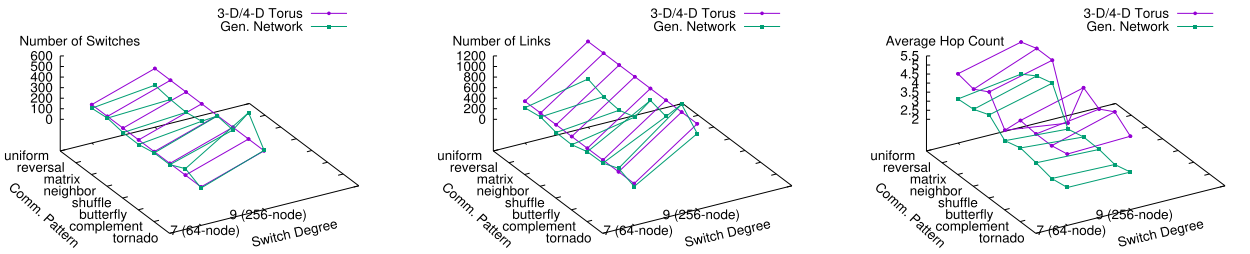
(switch and link) amount and average communication hop count. In addition, as the network size increases, such advantages become more significant. For comparison with 3-D/4-D tori topologies, we still specify the value of N in our generated topologies to be the same as counterpart 3-D/4-D tori topologies. From Fig. 12 (d) to Fig. 12 (f), we can see that the generated topologies outperform counterpart 3-D/4-D *tori* topologies for all communication patterns except for *complement* in terms of resource amount. For average hop count, the generated topologies obviously lead counterpart 3-D/4-D tori topologies in all cases. In other words, when our generated topologies allocate the same number of time slots as that in traditional tori networks, the number of occupied network resources including switches and links can be largely saved and the average communication hop count can be significantly reduced as well in any case.

Next, to investigate the tradeoff between N and network cost in our topology generator, we make a comparison with a standard mesh network. We use *uniform* as the communication pattern in 1024-node mesh networks and our generated 1024-node topologies.

Figure 13 shows the relative resource usage when setting different values of N in our generated topologies with a mesh network as baseline. We can see that a mesh network requires at least 14 slots, and when our generated topology consumes the same number of slots it only uses 54% switches and 47% links. If we reduce the value of N in the generated topologies, the relative resource usage increases accordingly since it uses a greater number of network resources. As depicted in Fig. 13, when $N = 5$, the number of used switches and links are comparable to that in a mesh network. In other words, if we use a equivalent number of switches and links in our generated topology to that in a mesh network, only 5 slots are required for each switch,



(a) # of switches (2-D torus) (b) # of links (2-D torus) (c) Avg. hop count (2-D torus)



(d) # of switches (3-D/4-D torus) (e) # of links (3-D/4-D torus) (f) Avg. hop count (3-D/4-D torus)

Fig. 12 Simulation results for comparison between tori and generated topologies by our design methodology.

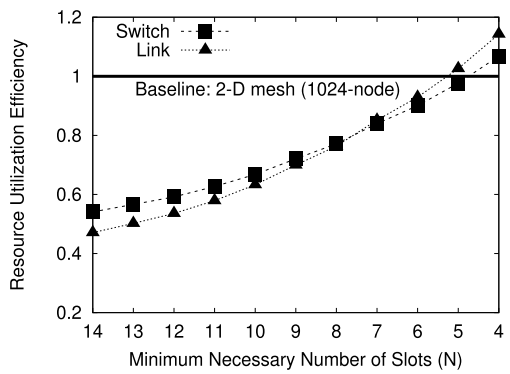
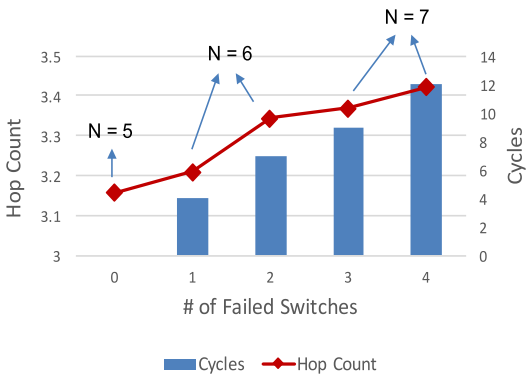
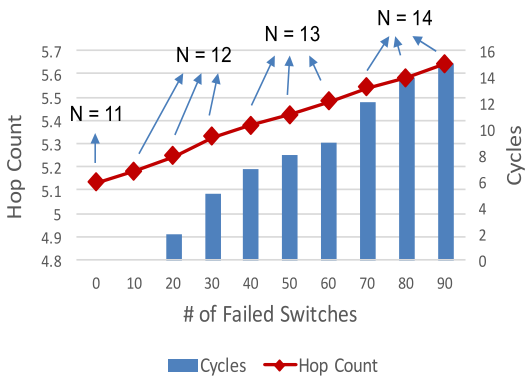


Fig. 13 Tradeoff between N and network cost (comm. pattern = *uniform*, # of nodes = 1024).



(a) 64 nodes



(b) 1024 nodes

Fig. 14 The average hop count and minimum extra cycles for topology reconfiguration in the network.

which is far less than 14 slots in a counterpart mesh network.

5.3 Topology Reconfiguration

We evaluate the performance of topology reconfiguration in our target CS network. The communication pattern is set to be *uniform* and the maximum switch degree is assumed to be 5 in the network.

Generally, the average hop count increases after topology reconfiguration because some node pairs could not take the shortest routing path due to the failure of one or several intermediate switches. A failed switch causes the permanent (hardware) failure of its directly connected links, through which any communication cannot pass. Thus, a failed switch can be seen as an equivalent to the failed links that are directly connected. There are also cases that the value of N increases after topology reconfiguration since the number of used time slots may exceed the current value of N. The minimum extra cycles for topology reconfiguration due to switch failure are shown as well. It can be seen that

more extra cycles are required for topology reconfiguration when more failed switches exist in the network. Comparatively, a larger size of network can tolerate more failed switches or links with no much performance degradation. For example, in a 64-node network (Fig. 14 (a)) if only one switch fails four extra cycles are required for topology reconfiguration, while in a 1024-node network (Fig. 14 (b)) if ten switches fail no extra cycles are appended for topology reconfiguration although the average communication hop count increases.

6. Conclusion

A major challenge to a circuit-switched (CS) network is how to support many concurrent flows efficiently to fully utilize link bandwidth. In this paper, we adopted a CS network for parallel computers applying statistical multiplexing of links to obtain guaranteed bandwidth and achieve low latency for each communication. The proposed circuit-switched network is a radical departure from current practice. We showed that it works efficiently for different types of communication patterns. We made optimization for link utilization and proposed a topology generator to dynamically generate a specific network according to a given communication pattern and the maximum switch degree in our target CS network. By performing a quantitative discrete-event simulation, we presented the advantages of performance and efficiency of a CS network generated by our design methodology. Beside performance benefits, it also provides a substantial reduction in network deployment cost. We consider that our target CS network can be a candidate design for next generation parallel computer networks.

Acknowledgments

This work is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] "Sb7700 series – switch-ib edr 100gb/s infiniband switch systems," http://www.mellanox.com/page/products_dyn?product_family=192.
[2] D. Fujiki, K. Ishii, I. Fujiwara, H. Matsutani, H. Amano, H. Casanova, and M. Koibuchi, "High-bandwidth low-latency approximate interconnection networks," in Proceedings - 2017 IEEE 23rd Symposium on High Performance Computer Architecture, HPCA 2017, IEEE Computer Society, pp.469–480, 2017.
[3] "Wikipedia: Circuit switching," https://en.wikipedia.org/wiki/Circuit_switching/.
[4] E. Chung, A. Nowatzyk, T. Rodeheffer, C. Thacker, and F. Yu, "An3: A low-cost, circuit-switched datacenter network," Tech. Rep., March 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/an3-a-low-cost-circuit-switched-datacenter-network/
[5] Y. Hu, T. Kudoh, and M. Koibuchi, "A case of electrical circuit switched interconnection network for parallel computers," in The 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT' 17), Taipei Taiwan, Dec. 2017.
[6] N. Farrington, G. Porter, S. Radhakrishnan, H.H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," ACM SIGCOMM Computer Communication Review, vol.40, no.4, pp.339–350, 2010.
[7] G. Wang, D.G. Andersen, M. Kaminsky, K. Papagiannaki, T.S.E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," ACM SIGCOMM, vol.40, no.4, pp.327–338, 2010.
[8] M. Kulkarni, M. Burtscher, C. Cascaval, and K. Pingali, "Lonestar: A suite of parallel irregular programs," in Proc. 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp.65–76, 2009.
[9] M. Koibuchi, H. Matsutani, H. Amano, D.F. Hsu, and H. Casanova, "A Case for Random Shortcut Topologies for HPC Interconnects," in Proc. of the International Symposium on Computer Architecture (ISCA), pp.177–188, 2012.
[10] I. Fujiwara, M. Koibuchi, T. Ozaki, H. Matsutani, and H. Casanova, "Augmenting low-latency hpc network with free-space optical links," in 21st International Conference on High-Performance Computer Architecture (HPCA), pp.390–401, Feb. 2015.
[11] K.J. Barker, A. Benner, R. Hoare, A. Hoisie, A.K. Jones, D.K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. Stunkel, and P. Walker, "On the Feasibility of Optical Circuit Switching for High Performance Computing Systems," in Proc. of ACM/IEEE Supercomputing Conference (SC), pp.12–18, Nov. 2005.
[12] G. Wang, D.G. Andersen, M. Kaminsky, M. Kozuch, T.S.E. Ng, K. Papagiannaki, M. Glick, and L. Mummert, "Your data center is a router: The case for reconfigurable optical circuit switched paths," in Proceedings of ACM Hotnets-VIII, New York City, NY, USA, Oct. 2009.
[13] L. Zhou, Z. Xu, X. Cheng, and Q. Huang, "An optical circuit switching network architecture and reconfiguration schemes for datacenter," Optics Communications, vol.335, pp.250–256, Jan. 2015.
[14] D. Wu, X. Sun, Y. Xia, X. Huang, and T.S.E. Ng, "Hyperoptics: A high throughput and low latency multicast architecture for datacenters," in 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), Denver, CO: USENIX Association, 2016.
[15] "Calient: The hybrid packet-ocs datacenter network final," http://www.calient.net/news/events/the-hybrid-packet-ocs-datacenter-network-final-12_29_12/.
[16] "Hybrid packet-optical circuit switch networks are new data center standard," http://www.datacenterknowledge.com/archives/2013/06/04/hybrid-packet-optical-circuit-switch-networks-are-new-data-center-standard/.
[17] M.C. Wu, S. Han, T.J. Seok, and N. Quack, "Large-port-count mems silicon photonics switches," in Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles, CA, USA, March 2015.
[18] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, Hong Kong, China, vol.43, no.4, pp.447–458, Aug. 2013.
[19] N. Farrington, A. Forencich, G. Porter, P.-C. Sun, J.E. Ford, Y. Fainman, G.C. Papen, and A. Vahdat, "A multiport microsecond optical circuit switch for data center networking," IEEE Photon. Technol. Lett., vol.25, no.16, pp.1589–1592, Aug. 2013.
[20] Y. Yin, R. Proietti, C.J. Nitta, V. Akella, C. Mineo, and S.J.B. Yoo, "Awgr-based all-to-all optical interconnects using limited number of wavelengths," in Optical Interconnects Conference, IEEE, pp.47–48, 2013.
[21] R. Proietti, C.J. Nitta, Y. Yin, V. Akella, and S.J.B. Yoo, "Tonak: A distributed low-latency and scalable optical switch architecture," in the European Conference on Optical Communicatoins (ECOC), London, UK, 2013.
[22] R. Proietti, Y. Yin, R. Yu, C.J. Nitta, V. Akella, C. Mineo, and S.J.B. Yoo, "Scalable optical interconnect architecture using awgr-based

tonak lion switch with limited number of wavelengths," Journal of Lightwave Technology, vol.31, no.24, pp.4087–4097, Dec. 2013.
[23] R. Proietti, Z. Cao, Y. Li, and S.J.B. Yoo, "Scalable and distributed optical interconnect architecture based on awgr for hpc and data centers," in Optical Fiber Communications Conference and Exhibition (OFC), San Francisco, CA, USA, March 2014.
[24] "Broadcasting (networking)," https://en.wikipedia.org/wiki/Broadcasting_(networking)/.
[25] S.Q. Moore and L.M. Ni, "The effects of network contention on processor allocation strategies," in Proceedings of the 10th International Parallel Processing Symposium, pp.268–273, 1996.
[26] Y. Yang and J. Wang, "Routing permutations with link-disjoint and node-disjoint paths in a class of self-routable networks," in Proceedings of the International Conference on Parallel Processing, pp.239–246, Aug. 2002.
[27] W.H. Ho and T.M. Pinkston, "A methodology for designing efficient on-chip interconnects on well-behaved communication patterns," in High-Performance Computer Architecture (HPCA), Anaheim, CA, USA, Feb. 2013.
[28] Q.-P. Gu and S. Pengo, "Wavelengths requirement for permutation routing in all-optical multistage interconnection networks," in Proceeding of the 14th IPDPS 2000, pp.761–768, May 2000.
[29] M. Koibuchi, K. Anjo, Y. Yamada, A. Jouraku, and H. Amano, "A simple data transfer technique using local address for networks-on-chips," IEEE Trans. Parallel Distrib. Syst., vol.17, no.12, pp.1425–1437, 2006.
[30] J. Duato, S. Yalamanchili, and L. Ni, Interconnection Networks: an engineering approach, Morgan Kaufmann, 2002.
[31] W.D. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2003.
[32] "Simgrid: Versatile simulation of distributed systems," http://simgrid.gforge.inria.fr/.

Yao Hu received the M.S. degree from Beijing University of Posts and Telecommunications, China, in 2009, and received the PhD degree from the Department of Computer Science and Engineering, Waseda University, Tokyo, Japan, in 2015. He is currently working as a project researcher in the National Institute of Informatics, Tokyo, Japan. His main research interests include the area of high-performance computing.

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 areas of high-performance computing and interconnection networks. He is a member of the IEEE and a senior member of the IEICE and IPSJ.