Packet Forwarding Cache of Commodity Switches for Parallel Computers

Shoichi Hirasawa*†
*National Institute of Informatics
† The Graduate University for Advanced Studies
hirasawa@nii.ac.jp

Hayato Yamaki ‡
‡ The University of Electro-Communications
yamaki@uec.ac.jp

Michihiro Koibuchi*†
*National Institute of Informatics
† The Graduate University for Advanced Studies
koibuchi@nii.ac.jp

***Abstract*—Switch delay dominates communication latencies in interconnection networks, especially for short messages because switch delays are massive relative to the link and packet injection delays. At a conventional switch, routing decision is based on CAM (Content Addressable Memory) table lookup, and it imposes a significant delay. Reducing the access latency to CAM is crucial for the upcoming low-delay switch in parallel computers. Besides the CAM latency problem, the packet forwarding rate is not proportional to the switching capacity on cutting-edge commodity switches of interconnection networks. A switch will not able to forward incoming packets at the maximum line rate. To resolve the latency and throughput problems, we explore an on-chip packet forwarding cache to a switch. An incoming packet avoids large-latency accessing a CAM forwarding table if the cache hits. It supports an almost 100% hit rate (no capacity miss nor conflict miss) for packets generated in up to 2K-node jobs. For 100% hit rate on larger jobs, we present a switchable hash function to refer to a packet forwarding table on a switch. The switchable hash function is optimized to typical network topologies, i.e., k-ary n-cubes, fat trees, and Dragonfly. The main idea is that a large number of packet destinations share a same index tag, resulting in the same required number of cache entries as the number of output ports. This design can be enabled by the path regularity of the above network topologies. Our evaluation results show that the reasonable packet forwarding cache supports a 933-Gbps line rate even for incoming shortest packets on the above network topologies. We illustrate that parallel applications obtain the performance gain of 5.07x speed up using the cache switches since the impact of the switch delay and link bandwidth is significant on the end-to-end communication performance.**

I. Introduction

Commodity interconnection networks, e.g., InfiniBand and Ethernet, are a mainstream of supercomputer's communication platform. As of June 2020, 80% of Top500 supercomputers use them [1]. A commodity interconnection network usually supports arbitrary network topologies by updating a packet forwarding table at a switch.

At a conventional commodity switch, routing decision is based on CAM (Content Addressable Memory)-based table lookup. A main problem of the packet forwarding processing is the significant operation latency to the CAM at a switch. The next-generation High Performance Computing (HPC) systems are to achieve low end-to-end latencies, e.g., under 1 μs across an exascale system [2]. This objective is partially achieved thanks to decreasing switch delays because switch delays are massive relative to the link and packet injection delays. Currently, Mellanox QM8700 achieves 130 ns switch delay, and some Mellanox switches achieve less than 100 ns delays. Since a conventional switch uses off-chip CAMs for forwarding packets, its access latency would reach dozen of nanoseconds that should be reduced.

Another problem is the low packet forwarding rate on a switch. Switching capacity (giga bits per second) is proportional to the aggregate line rate on a switch, e.g., 25.6 Tbps for 400Gbps $\times$ 64 ports in a cutting-edge switch [3]. Link bandwidth continues to increase to 1.6 Tbps in the 2020s, as illustrated in Ethernet Roadmap 2020 [4], and the supercomputer interconnect will follow the link-bandwidth roadmap. By contrast, packet forwarding rate (billion packets per second) using TCAM (ternary CAM) lookup is not much improved year by year because the operation latency of product TCAMs does not much decrease in recent ten years, e.g., six nanoseconds for Renesas Electronics R8A20410BG in 2009, while four nanoseconds for the successor R8A20611BG in 2021 [5]. Ethernet requires to perform packet forwarding processing every 84 Bytes at an input port because the minimum frame length and inter-frame gaps are 64 and 20 Bytes, respectively. When a single-port TCAM is located at each input port, the upper bound of an input-port throughput is 168 Gbps (= 1/4ns $\times$ 84 Bytes $\times$ 8). Maximum transfer unit (MTU) is 1,518 Bytes in Ethernet. If an inter-process message length becomes longer than 1,518 Bytes, it is divided and transferred by multiple Ethernet frames. When all frames have the longest length (1,518 Bytes), the packet forwarding processing achieves up to 3.0 Tbps line rate (=1/4ns $\times$ 1,518B $\times$ 8). This is the theoretical upper bound of the line rate. In practice, various frame sizes should be burstly proceeded. For example, when average frame length is 1 KiB, the line rate supported by the packet forward processing is up to 2.0 Tbps (=1/4ns $\times$ 1KiB $\times$ 8). Since over 2 Tbps link will appear shortly, the packet forwarding processing should be much improved. Consequently, the gap between a line rate and a packet forwarding rate will become a serious bottleneck in parallel computing.

To mitigate the throughput gap, an existing straight-forward solution is to operate parallel accesses to multiple TCAMs or their multi-port extension on a switching ASIC [3]. Of course,

this parallelism approach will be unrealistic due to the vast cost as the line rate increases.

To improve the operation latency to compute routes and the packet forwarding rate, we explore an on-chip packet forwarding cache to a switch in this study. An incoming packet avoids accessing a TCAM forwarding table if the cache hits. Only an exclusive layer-1 (L1) cache at an input port contributes to achieving a high line rate, e.g., 800 Gbps for the incoming short packets. An L2 cache access latency becomes a few nanoseconds (e.g., 2ns) that do not provide a high line rate, i.e., higher than 336 Gbps (= 1/2ns $\times$ 84 Bytes $\times$ 8). The L1 cache size is strictly limited by the chip area. Unfortunately, the baseline cache switch would lead to a low hit ratio close to *zero* when the network size is large (see Section IV). In this study, we present a switchable hash function to index a packet forwarding cache on a switch for almost 100% hit rate for packets generated within large jobs on k-ary n-cubes, fat tree, and Dragonfly. It completely removes the capacity and conflict cache miss regardless of the network size.

This work makes the following contributions:

- We illustrate a packet forwarding cache architecture on a switch that, on most jobs on arbitrary network topologies, 2K-cache entries with a four-way set-associative provide almost 100% cache hit rate (no conflict nor capacity miss). (Section III)
- We demonstrate an additional switchable hash function on the packet forwarding cache architecture for elephant-nose large jobs on some specified network topologies. We state some typical configurations of the hash function. It requires taking a custom packet destination addressing on k-ary n-cubes, fat trees, and Dragonfly. (Section IV)
- Our evaluation results show that the packet forwarding cache provides up to 933-Gbps line rate on the above network topologies. We illustrate that parallel applications obtain the gain of 5.07x speed up using the packet forwarding cache. (Section V)

The remainder of the paper is organized as follows. Section II describes background information. Section III describes the switch organization using the forwarding packet cache. Section IV describes the switchable hash function on the proposed switch. Section V describes the performance evaluation of the packet forwarding cache with the switchable hash function. Section VI discusses alternative ways to improve the hit rate of the packet forwarding cache. Section VII concludes with a summary of our findings and perspectives on future work.

II. Background Information

A. Typical Network Topologies

A few network topologies are traditionally used to interconnect compute nodes in parallel computers. Fat tree, torus, and Dragonfly are used in most top 30 supercomputers as of Nov. 2019 [1]. Routing on these network topologies includes a strong regularity that helps design a routing algorithm, e.g., dimension-order routing on tori. A few supercomputers implemented network topology and routing by custom switching chip of supercomputers, e.g., BlueGene/Q, and they can implement hardware synthesis using the target regularity as a routing logic. By contrast, the remaining interconnection networks rely on commodity technologies, e.g., InfiniBand or Ethernet, that support arbitrary network topologies. Even at L2 Ethernet, VLAN technology simply implements network topology that includes loops [6], [7]. Supporting arbitrary network topology is enabled by forwarding tables at a switch on an L2 Ethernet switch. InfiniBand is frequently used for building typical network topologies, e.g., fat trees with director-and-ToR switches on Summit, Sierra, and Frontera. In fact, the commodity interconnection technology is mainly used in some typical network topologies though it supports arbitrary network topologies.

In this study, our packet forwarding cache targets a commodity switch that supports arbitrary network topologies, as Ethernet and InfiniBand interconnects. For the above typical network topologies, we additionally provide special hash functions for almost 100% cache hit rates.

B. High-Throughput Switching ASIC

Hyperscale datacenters highly demand a top-of-rack high-throughput single-chip switch. Broadcom releases the design of Tomahawk 3 Ethernet switching ASIC (12.8 Tbps) in 2018, and Tomahawk 4 (25.6 Tbps) in 2019. It is expected that a switch ASIC will reach 51.2 Tbps in the first half of the 2020s. The application of Tomahawk switching ASIC to the HPC domain is a Rosetta switch in Cray Slingshot interconnection network which uses Dragonfly network topology. Intel and Barefoot provide a 12.9 Tbps P4-programmable Tofino2 switch, while Rockley Photonics also illustrates a hyperscale switch [8]. Continuing to increase the switching capacity of a switching ASIC relies on optical integration.

Besides the switching capacity, the packet forwarding rate is far from a line rate when incoming packets are burstly short in cutting-edge switch products. For example, a Tofino2 switch provides 12.8 Tbps (400 Gbps $\times$ 32) switching capacity while it has only 6 Bpps (billion packets per second) that leads only 4-Tbps throughput (6 Bpps $\times$ 84 Bytes $\times$ 8) for a minimum packet length. Similarly, Arista 7060X series provides 12.8 Tbps with 3.3 Bbps to 9.52 Bpps. A typical packet forwarding engine relies on TCAM. Intel Tofino2 switching chip is implemented on 2.5D chiplets, and it seems that TCAMs are located outside the switching chip. The bottleneck of the forwarding packet rate is the access latency within TCAM regardless of its location (on-chip or off-chip), and TCAM itself almost maintains its read latency year by year, as described in Section I. The gap between switching capacity and forwarding packet rate will increase even when large TCAM will be integrated into 2.5D chip implementation.

It is difficult to provide the proportional packet forwarding rate to a high line rate on a future switch even for long packets, as described in Section I. The key design to resolve the problem of the packet forwarding performance is a packet forwarding cache architecture explored in this study.

C. Cache Architecture at Internet Routers

Cache architecture to forwarding packets has been discussed mainly for Internet routers. The prior studies concern either high hit rate or longest prefix matching.

1) High-throughput Cache: In the 1990s, the routing table is stored in DRAMs whose access latency becomes some hundreds of nanoseconds. The fast-path technology bypasses the original datapath, and it uses cache whose access latency is some nanoseconds to improve the router latency. The memory structure is similar to that in a traditional processor. Since TCAM and SRAM are magnitudely expensive and power-hungry than DRAM, a cache strategy using a small amount of special-purpose memory is reasonable. The cache replacement algorithms, such as least recently used (LRU) and least frequently used (LFU), are evaluated on digital subscriber line (DSL) or ISP's traffic patterns [9]. A router-specific cache replacement algorithm is proposed called traffic-aware flow offloading (TFO) [10]. The TFO uses traffic statics over multiple-time scales and leverages Zipf's law. When a cache hits, incoming traffic bypasses the controller in a router model consists of the forwarder and controller [10].

In the 2010s, a TCAM can store all routing-table entries for improving the packet forwarding rate. Currently, the access latency to the table entries in a TCAM is not proportional to the switching capacity in a cutting-edge switch, as described in Section I. In this context, inserting an on-chip SRAM cache is investigated in [11]. If cache hits, avoiding accessing a TCAM leads high packet forwarding rate. In this study, we vote for the on-chip cache strategy to use our baseline cache structure in a baseline switch.

2) Software-Defined Network: Software-defined networks would require a larger number of rule tables than a number of table entries in modern TCAMs [12], [13]. Accessing to the controller introduces slow packet forwarding, but the cache strategy should care about the rule-dependent analysis for the longest prefix match. If the cache does not include the longest prefix entries, misrouting may occur with the shorter prefix matching at a cache. Fortunately, the parallel computing system does not use complex rules, and we are free from the rule consistency problem at a packet forwarding cache.

To the best of our knowledge, there are no prior works on routing cache on parallel computer systems. We should investigate its impact on the performance of parallel applications.

D. Traditional Switch Microarchitecture

We describe our baseline switch microarchitecture. A four-stage router quoted from [14] is assumed as a baseline switch in this paper as shown in Figure 1. In the router, a header flit is transferred through four pipeline stages that consist of the routing computation (RC) stage, virtual channel allocation (VA) stage for output channels, switch allocation (SA) stage for allocating the time-slot of the crossbar switch to the output channel, and switch traversal (ST) stage for transferring flits through the crossbar. An example of a four-flit packet transfer is illustrated in in Figure 2.

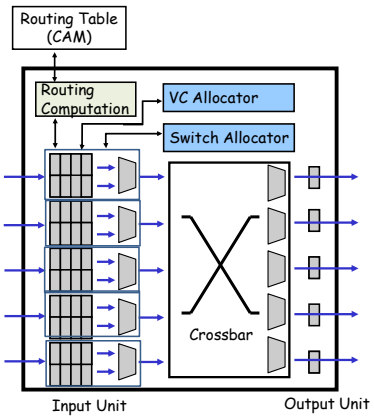


Figure 1. Block Diagram of Traditional Switch.

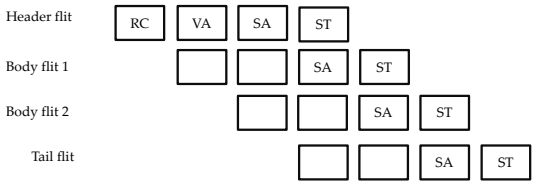


Figure 2. Pipeline Processing of a Packet.

Indeed, each of the four operations is executed with multiple clock cycles. Existing commercial high-performance switches are usually a black-box for users, and their pipeline structures are partially opened; an InfiniBand DDR switch has 140 nanoseconds delay, while Fujitsu 10GbE switches take 450 nanoseconds delay (312.5 MHz, 140 cycles). RHiNET-2/SW and RHiNET-3/SW and their pipeline structures include additional stages, such as ECC decoder and encoder.

Although the parallelism of control dependency on the pipeline structure can be logically relaxed by look-ahead routing and speculation [15], actual switch products still have over 100 nanoseconds delay.

E. Job Size of Parallel Computers

A parallel computer typically assigns a job to a subset of compute nodes that are not overlapped to a subset occupied by another job. When a switch stores all the table entries used in the job, we achieve *zero* capacity miss at the packet forwarding cache.

We analyze the job sizes of some modern supercomputers, and their CDF is plotted in Figure 3. We used Parallel Workloads Archive (PWA) [16] as an input trace.

The legend identifies the number of processors, and the vertical axis represents the number of processors that are not the number of compute nodes. For example, in RICC, a single compute node consists of eight processors, and CEA-Curie includes both CPU and GPU that leads different numbers of processors on a compute node. Thus, each job occupies a smaller number of compute nodes than that in Figure 3.

Figure 3 suggests that RICC executes 98% of jobs that

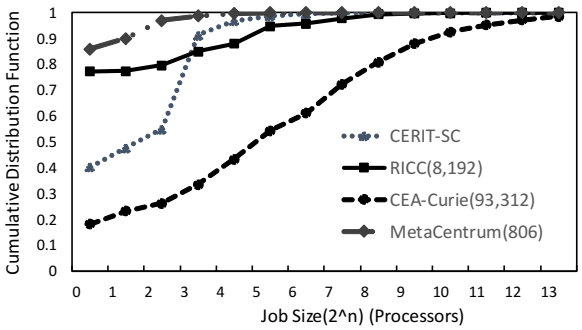


Figure 3. The Cumulative Distribution of Parallel-Job Sizes.

occupy up to 32 compute nodes. Most jobs are small on average, and 2*K* table entries are enough for them. It is reasonable to support such job size at the cache on other network topologies.

Notice that this study aims at enabling high-bandwidth low-latency interconnection networks for any job size. This analysis results in Section III indicates that 2*K* table entries are enough for most jobs in a switch. However, the remain elephant-nose (large) jobs, that would be important in the HPC domain, suffer from the low cache hit rate on the cache switches. form an arbitrary topology. The switchable hash function resolves the problem of the low cache hit rate when typical network topologies are used.

III. Packet Forwarding Cache

A switching ASIC should be widely used in parallel computers and datacenters. We explore the packet forwarding cache design for arbitrary network topologies, like InfiniBand and Ethernet.

A. Routing-Computation Unit

Figure 4 illustrates our target switch with packet-forwarding cache. Later, we call it the cache switch. It is the same as the baseline switch in Figure 1 except for the routing-computation unit.

After an incoming packet is stored at an input port, its output port information is obtained via a CAM table lookup in the traditional switch.

By contrast, in the cache switch, an incoming packet accesses a packet-forwarding cache. A referred cache line is identified by a hash value computed by a destination node information stored in a packet header. If tag information of the referred cache line matches the packet's destination, this is a case of cache hits, then its output port is obtained from the cache line. If the cache misses, a CAM table lookup is performed. If the switch hits a cache for an incoming packet, the time to complete the routing computation decreases by avoiding a CAM access.

B. Cache Architecture

1) Cache Hierarchy: Multi-layer data cache has been commonly used in a chip multiprocessor. However, we do not use

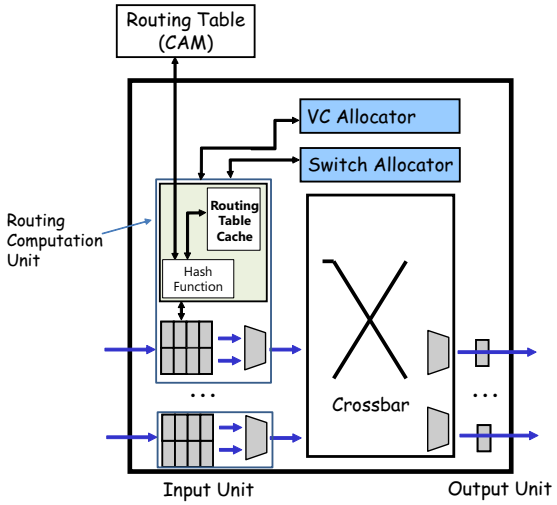


Figure 4. Block Diagram of Switch with Routing-Table Caches.

a multi-layer cache because only the exclusive layer-1 cache can contribute to the high-throughput line rate. Our CACTI simulation results showed that a conventional layer-2 cache has a few nanoseconds access latency. Even if a two-stage pipeline processing is supported to access an L2 cache, the bandwidth is limited to 672 Gbps throughput (1/2ns $\times$ 2 pipelines $\times$ 84 Bytes $\times$ 8) for short incoming packets.

2) Consistency Model: Unlike a processor's cache, we do not have to update the routing information frequently. The update of the routing information may occur when faulty hardware occurs in an interconnection network. In such a case, simple control that cleans all the table entries once is practical, and a consistent operation for data hazard, e.g., WAW (Write After Write), RAW, and WAR, becomes a trivial problem.

3) Number of Cache Ways and Size: We analyze the ratio of entries replaced by conflicts under different associative numbers of caches in Figures 5 and 6. Figure 5 illustrates the conflict miss rate when a packet destination is uniformly distributed at an input port of a cache switch. Assume that a cache has 2*K* table entries. Figure 6 illustrates the conflict miss rate with real benchmark application workloads.

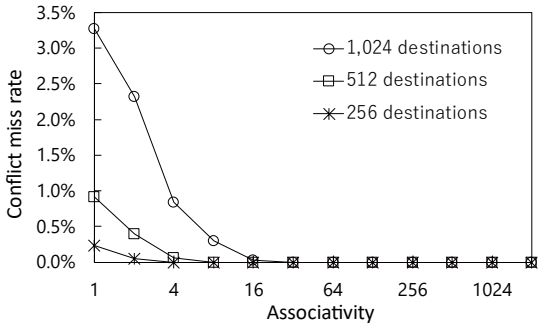


Figure 5. Associative Number and Ratio of Entries Replaced by Conflicts (Uniform Traffic, 2*K* Table Entries).

The legend represents the number of packet destinations.

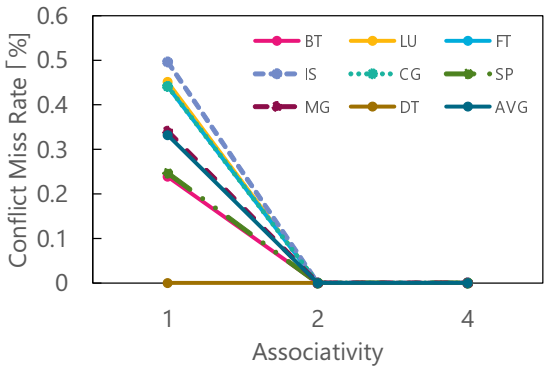


Figure 6. Associative Number and Ratio of Entries Replaced by Conflicts (NAS Parallel Benchmarks).

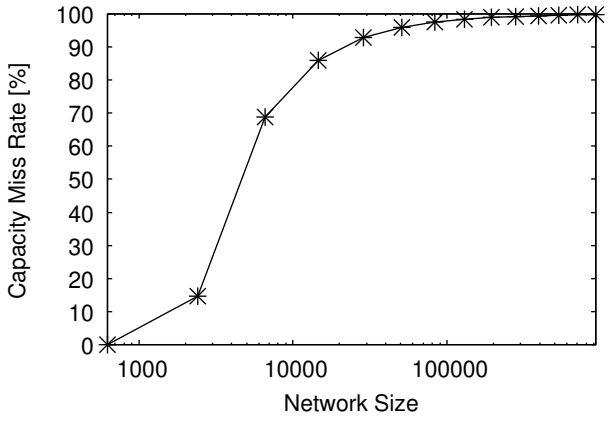


Figure 7. Capacity Miss Ratio on k-Ary 4-Tori (Uniform Traffic, 2K Cache Entries).

The results illustrate that the four-way set-associative cache is reasonable for almost avoiding the conflict cache miss.

We also analyze the conflict miss rate when NAS Parallel Benchmarks are executed on the system condition with 3-D torus (8×8×4). It also suggests that the two- and four-way cache avoid the conflict cache miss.

We analyzed the relationship between the number of ways and cache hit rates for reasonable job sizes of real parallel computers. Our finding was that most jobs used less than $2K$ nodes. Using $2K$ table entries decreases the possibility of capacity miss of a cache.

Another finding was that four-way set-associative cache introduces almost *zero* conflict miss on the reasonable scenario. Later, we set $2K$ table entries with four-way set-associatives as baseline in the cache switch.

IV. Switchable Hash Function

The heart of the packet forwarding cache is to support the switchable hash function. The switchable hash function enables to provides 100% cache hit rate except for the initial miss even for large jobs on typical network topologies.

A. Problem Statement

We quantitatively illustrate a serious low cache hit rate as the system size becomes large. We assume four-dimensional tori with a dimension-order routing, and random traffic in which each node exchanges a message to a randomly selected destination node. Since parallel program sometimes generates all-to-all communication, the random traffic should be considered. Cache misses are classified into initial, capacity, and conflict. As the job size increases, the capacity miss is a crucial problem. Figure 7 illustrates the relationship between the job size and the capacity miss rates of the cache. Surprisingly, we found an average low hit ratio close to *zero* when the network size is large.

B. Assumption and Our Approach

We explore a packet forwarding cache to address the problem of the low hit ratio close to *zero*. A packet address takes a wide range, e.g., 24 bit, in interconnection networks. A hash function converts a (short) cache tag from a packet address before the table lookup. A fast typical hash function takes a Cyclic Redundancy Check (CRC) to distribute the tag uniformly.

To achieve almost 100% cache hit rate, our approach relies on a topology regularity. Existing parallel computers usually use k-ary n-cubes, fat tree, and Dragonfly network topology. We target to support almost 100% hit rate only when they are used.

Our idea is to provide multiple hash functions optimized to the above network topologies. The hash function assumes to use its custom node address. For example, the custom node address can be set as local identifier (LID) in InfiniBand. The procedure to use a switchable hash function is described as follows.

1) Each compute node is assigned to its custom address.
2) Each switch fills up CAM entries when a parallel computer is deployed.
3) All cache entries are flashed. Each switch then sets a suitable hash function before a job is executed, and turns on a hash.

When an incoming packet arrives, a cache tag corresponding to the destination address is obtained by the hash function. To easily maintain the consistency of CAM and cache, the switchable hash function has to output the index tag that is a key of cache.

We prepare four datapaths on the hash function in Figure 8, k-ary n-cube, fat-tree/Dragonfly, arbitrary topologies, and link aggregation (LAG). The datapath for arbitrary topologies follows a CRC computation, and we follow the CRC design in [11]. The function to select a link among LAG is performed in parallel if LAG is used. The remaining two datapaths are stated as follows.

C. K-Ary N-Cube

We state a hash function to k-ary n-mesh/torus in Algorithm 1. We assume that dimension-order routing is used. Each compute node is assigned to the n-dimensional coordinates

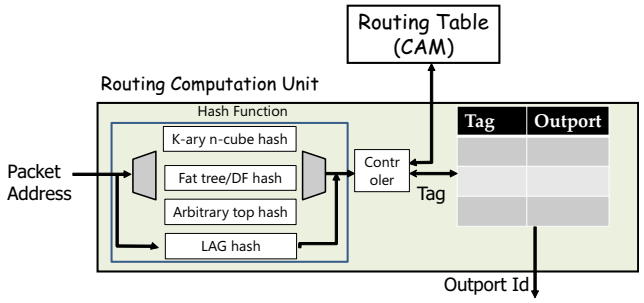


Figure 8. Routing Computation Unit.

Algorithm 1 Hash function on k-ary n-cube.

Input: Coordinates of current switch node $(c_{n-1},\ldots,c_0)$ and destination compute node $(d_{n-1},\ldots,d_0)$.
Output: Cache index tag.

```
1: procedure :
2:     for i = 0 to n − 1 do
3:         X_offset := d_i − c_i;
4:         if x_offset > ⌈k/2⌉ then
5:             tag := X_{i,a};
6:         else if x_offset > 0 then
7:             tag := X_{i,+};
8:         else if x_offset < −⌈k/2⌉ then
9:             tag := X_{i,b};
10:        else if x_offset < 0 then
11:            tag := X_{i,−};
12:        end if
13:    end for
14:    tag := Local ComputeNode;
15: end procedure
```

$(X_{n-1},\ldots,X_1,X_0)$, and each switch has compute nodes. Each output-port information in the cache corresponds to a link. On tori, the wraparound channel makes $X_{i,a}$ the $X_{i,-}$ direction in Algorithm 1. On meshes, $X_{i,a}$ represents the $X_{i,+}$ direction. Similarly, $X_{i,b}$ makes the $X_{i,+}$ and $X_{i,-}$ directions on tori and meshes, respectively.

We minimize the required number of table entries, that is the number of ports on a switch. Multiple destinations that use a same output link refer the same table entry. For example, the packet destinations (0,2), (1,2), and (2,2) share the same table entry ($X_{0,+}$) at the switch (1,1) in Figure 9. In Figure 9, each output port is marked as $X_{0/1,+/-}$, that is the table entry. We fully utilize the regularity of dimension-order routing on k-ary n-cubes to minimize the number of table entries.

A high-radix switch sometimes uses multi-rail links, also known as link aggregation (LAG). The link selection within a LAG can be performed by the CRC hash function, and we omit it in Algorithm 1. Each table entry in Algorithm 1 should be stored so as to avoid the conflict miss on n-way associative cache. We omit the detail of an allocation to avoid conflict miss in this study because its allocation is obvious.

D. Fat Tree and Dragonfly

Up*/down* routing is assumed in fat trees. Assume that a switch at the layer i has n upper and down links. Recent parallel computers use a fat tree that consists of some high-

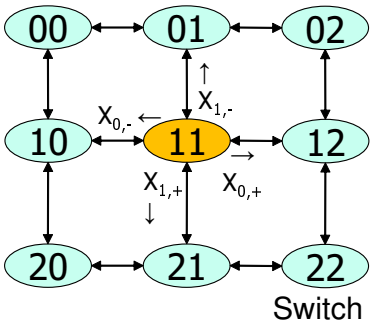


Figure 9. An Example of 3-Ary 2-Mesh.

Algorithm 2 Hash Function on n-Dimensional Fat Tree.

Input: Coordinates of current switch node (dim, $c_{n-1},\ldots,c_0$) and destination compute node $(d_n,\ldots,d_0)$.
Output: Cache index tag.

```
1: procedure :
2:     for i = n − 1 to dim do
3:         if d_{i+1} ≠ c_i then
4:             tag := up_{d_i};
5:         end if
6:     end for
7:     tag := down_{d_i};
8: end procedure
```

radix director switches and many ToR switches. The hash function supports such a high-radix fat tree.

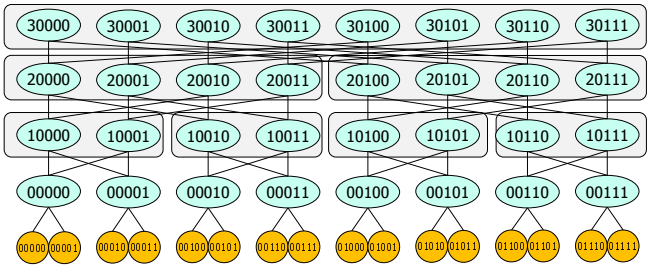


Figure 10. An Example of 2-Ary 4-D Fat tree. (n=4, k=2)

Algorithm 2 represents the hash function on a fat tree. Each switch node is assigned to the $(n+1)$-dimensional coordinates (dim, $c_{n-1},\ldots,c_0$), where dim represents the dimension number and $c_j\{0 \le j < n, 0 \le c_j < k\}$ for the address in the dimension. Each compute node is assigned to the $(n+1)$-dimensional coordinates $(d_n,\ldots,d_0)$. Each output port information in the cache corresponds to a link. It is identified by up_j and $down_j$ for $0 \le j < k$. The links up_0 and $down_0$ are connected to the left-most switch of neighboring up/down layers. Figure 10 represents an example of the node addresses in Algorithm 2.

The Dragonfly is a meta-topology that states a geometric (intra and inter-rack) network topology. A typical configuration takes a densely connected intra-rack network topology and a virtual router that consists of switches on a single rack takes a densely connected inter-rack network topology. Such a typical configuration of the Dragonfly network topology can work with the hash configuration of fat tree in Algo-

rithm 2. We illustrate the Dragonfly in Figure 11 in which the inter- and intra-rack interconnection networks are defined. The upper most layer of switch nodes consist the inter-rack interconnection network, while the remaining layers of switch nodes consist multiple intra-rack network, intra-racks 0 to 7 in Figure 11. The links connected to the different coordinates of different layers (up/down) in fat tree topology are physically connected by design to a switch node in the same layer. As a result, the connected switch nodes in the same coordinates form a intra-rack group. With all groups, the network consists the Dragonfly, in which the hash function of fat tree also works for each switch node.

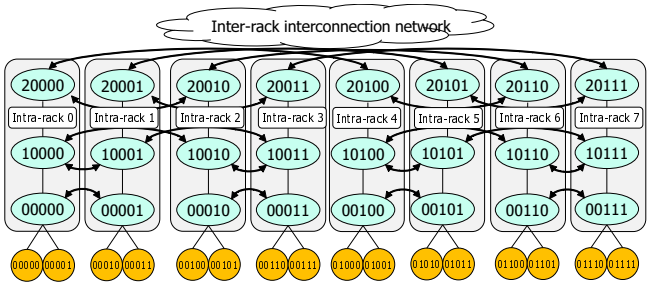
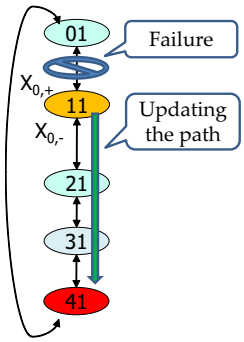


Figure 11. Dragonfly corresponding to the fat tree in Figure 10

E. Flexibility and Limitation

A same hash function is shared by tori and meshes. It can support some routing functions by updating the table entry of the packet forwarding cache. For example, when the $X_{0,+}$ link of node 11 fails permanently, updating the cache entry avoids the faulty link is illustrated in Figure 12.



Routing Table Cache @ Switch 11

Tag (old)	Output port
X0,a	X0 - Dir
X0,+	X0 + Dir
X0,-	X0 - Dir
X0,b	X0 + Dir

Tag (new)	Output port
X0,a	X0 - Dir
X0,+	X0 - Dir
X0,-	X0 - Dir
X0,b	X0 - Dir

Figure 12. An Example of a Faulty 5-Ary 1-Cube.

We give up to support 100% hit rate on the other network topologies. Supercomputers and datacenters historically use some network topologies supported by the packet forwarding cache. However, if the trend of the network topology changes, i.e., when a new network topology appears, the packet forwarding cache works as a common four-way set-associative, which may lead low hit rate in a large parallel computer. Another option is to add a hash function optimized to a newly introduced network topology to a cache switch by the hardware remake, though it may be costly. This is the limitation of this study, though the possibility to change the trend of the network topology is low.

V. Evaluation

We target a 64-port switching ASIC (64 $\times$ 800 Gbps). We ultimately evaluate the packet forwarding cache in terms of chip area, latency, and application performance.

A. Cache Size and Latency

We estimate the cache area and latency using the CACTI v6.5 simulation, to guarantee that the packet forwarding cache is feasible in terms of the area overhead and the forwarding throughput. The transistor model is a high-performance 45nm process. We assume four-way set-associative.

Figure 13 illustrates the area overhead of the cache. Since our target is a 64-port switch, the total cache overhead area becomes 14.6mm^2 for 2,048 entries (0.228mm^2 in the graph plot) in a chip. Since switching ASICs have large area, e.g., 329 mm^2 in Tofu of K-computer ICC (65nm technology), we consider that 2K entries for a 64-port switch are acceptable.

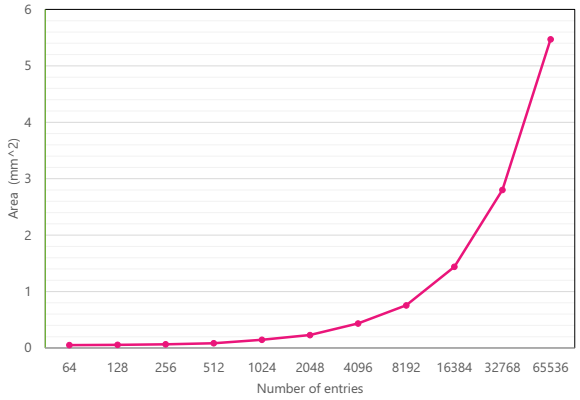


Figure 13. Cache Area Overhead at an Input Port.

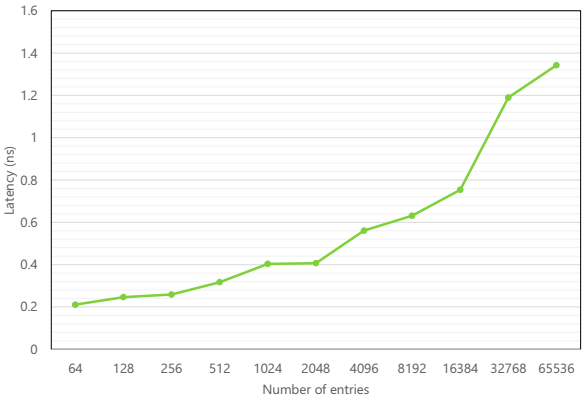


Figure 14. Cache Latency Overhead at an Input Port.

Figure 14 illustrates the latency overhead of the cache, and it does not include the switchable hash function. When 2,048

entries are used, the latency is 0.41 nanoseconds. Ideally, the packet forwarding throughput becomes 1.64 Tbps (1 / 0.41 $\times$ 84 $\times$ 8) when the switchable hash function is not included. We conclude that the packet forwarding cache is able to achieve an 800-Gbps line rate with a moderate area overhead (if cache hits).

B. Switchable Hash Function

1) Condition: The switchable hash function is designed with the Nangate 45nm Open Cell Library [17]. We completed its synthesis using Synopsys Design Compiler O-2018.06-SP4.

Routing information is set to 24 bits for supporting up to $2^{24} = 16M$ addresses. We omit a datapath for the CRC cache function for arbitrary network topologies because we can borrow the results from the design in [11]. It is essentially needed even if the switchable hash function is not used in a cache switch. Notice that the results illustrated the area overhead of the CRC hash function is trivial compared to area of a modern switching ASIC.

In *k*-ary *n*-cubes, our design supports the following configuration, 256-ary 3-cube, 64-ary 4-cube, 16-ary 6-cube, 8-ary 8-cube, and 4-ary 12-cubes. Since we do not have to use all the node addresses to the above configuration, the hash function supports any network topologies that can be embedded in them. For example, it is possible to construct a 4-ary 12-cube in which two links are bundled as a link aggregation in a 64-radix cache switch,

In fat trees, the oversubscription should be carefully set. It is the ratio of the number of lower links against the number of upper links at an intermediate switch, and we support 1 : 1. It is obvious that the hash function to support 1 : 1 oversubscription can embed an $n : 1$ fat tree, and a Dragonfly that supports a fully connected intra and inter-rack network topology by the configuration of the 3-dimensional fat tree.

Another CRC computation is performed to support LAG. We support up to a 16-link bundle (4 bits), and a link is selected from the input packet's destination address (24-bit) with the CRC computation.

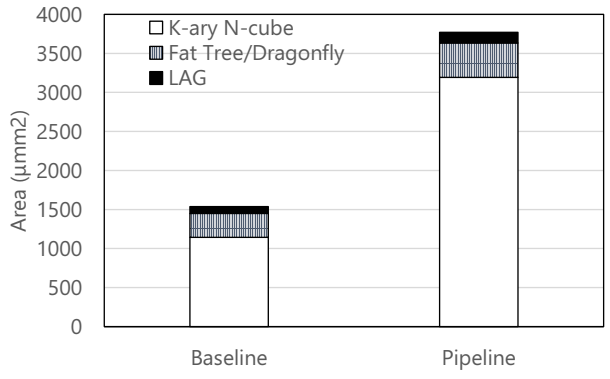


Figure 15. Hash-function Area Overhead at an Input Port.

2) Results: Figures 15 and 16 illustrate the area and latency overheads of the switchable hash function, respectively. We

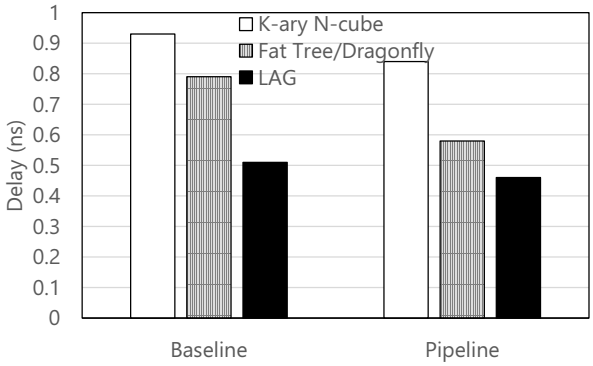


Figure 16. Hash-function Latency Overhead at an Input Port.

design two versions (combinational and sequential circuits), and they should be selected according to the latency requirements. The baseline represents the combinational circuits. The pipeline represents the sequential circuits, and two cycles are needed for enabling higher operating frequency. A port selection among a bundled link is selected by the packet destination, and a CRC circuit is used as shown in Figure 8 for the link aggregation.

We support n-cubes where n is 3, 4, 6, 8, and 12 in this evaluation. If k is limited to a specific value, e.g. 6, the area can be almost one four, i.e., $300\mu mm^2$ in the baseline and $352\mu mm^2$ in the pipeline. (we omit these values in the figure.) Similarly, if only a 5-dimensional fat tree is supported, the area overhead becomes $114\mu mm^2$ and $165\mu mm^2$ on the baseline and the pipeline, respectively. The operation of the hash function for fat trees is based on a bitwise comparator, and it becomes lightweight compared to that for k-ary n-cubes. Thus, The hash function for k-ary n-cubes occupied 85% of the total amount of the hardware for the switchable hash function.

Although there is a trade-off between generality and area overhead, its area overhead is small compared to a switching ASIC, e.g., $329mm^2$ of ICC chip in the K computer (10 ports, Fujitsu Semiconductor 65nm process technology). In this context, we conclude that the area overhead of each hash function is trivial even if we support various configurations of fat tree, k-ary n-cubes and Dragonfly.

When a network topology is fixed, its switchable hash function and LAG computational function are done in parallel. Thus, the latency can be regarded as the maximum value of the switchable hash function and LAG computational function. It is less than 1 nanosecond. If we support the above limited configuration, the latency further decreases. For example, when only 6-cubes are supported, the latency becomes 0.54ns and 0.47ns on the baseline and the pipeline, respectively. If three-stage pipeline is used, each stage can be completed within most 0.72ns. The access is fully pipelined with three stages, the packet forwarding throughput becomes 933Gbps (1/0.72ns $\times$ 84 $\times$ 8) for incoming shortest packets.

We conclude that the latency overhead would not be a severe bottleneck compared to the switch latency, i.e., some dozens

of nanoseconds, and we can support typical configuration for k-ary n-cubes, fat trees, and Dragonfly.

C. Parallel Application Performance

1) Condition: The ultimate metric is application performance. The purpose is to illustrate the impact of the low latency of the cache switch on the application performance. We use discrete-event simulation to evaluate the performance of parallel application benchmarks. To this purpose, we use the SimGrid simulation framework (v3.12) [18] to simulate the execution of unmodified parallel applications that use the Message Passing Interface (MPI) [19]. The parameters are listed in Table 1.

Table 1. Parameters of SimGrid Simulation.

	Case 1	Case 2
RC delay (w/o cache)	10 ns	10 ns
RC delay (w cache)	1 ns	1 ns
Other switch delay	50 ns	50 ns
Compute power	100GFlops / core	100GFlops / core
Network topology	4-D torus	Dragonfly
Number of Nodes	256 switches	256 Switches
	4^4	16 Groups $\times$ 16 nodes

The routing-computation (RC) delay is set to 1 nanosecond if the cache hits. As the results in the previous subsection, the cache latency is 0.41ns and the delay of the switchable hash function is 0.72ns, thus 1ns in total. The RC delay is set to 10ns, including 4ns delay within a TCAM in the conventional switch.

A conventional switch that uses CAM assumes to use a 200Gbps, 400Gbps, 800Gbps, and 1.6Tbps link. 1.6Tbps is used for simulating link aggregation. It would not guarantee 400Gbps and higher line-rate throughput for incoming shortest packets. However, in this evaluation, for the sake of simplicity, we assume that every switch works to support line-rate throughput.

We simulate the execution of the MPI NAS Parallel Benchmarks version 3.3.1 [20] (Class B for BT, CG, DT, LU, MG and SP, and Class A for FT and IS benchmarks), the matrix multiplication example provided in the SimGrid distribution (MM), and the Graph500 benchmark version 2.1.4 [21].

2) Simulation Results: Figures 17 and 18 show performance results for the baseline and the cache switches, normalized to the performance achieved by the baseline switch using 100Gbps links. The *y*-axis represents the relative operations per second (Mop/s for NPB benchmarks), and the higher value is better.

The cache-switch delay is 51ns, while the conventional switch takes a 60ns delay. The delay difference results in the performance improvement by the cache switch. The network diameter of the 4-D torus is 8 hops while that of the Dragonfly is 3 hops. The impact of the switch delay is smaller in the Dragonfly.

An important performance factor is the link bandwidth. As the link bandwidth increases, the application performance drastically improves, especially for DT by 15.3x and 15.7x, of 4-D torus and Dragonfly respectively. This result interestingly illustrates the demand of the network bandwidth. On average, 5.07x maximum performance improvement is achieved on 4-D torus and 4.27x on Dragonfly. As the total link number of Dragonfly is 3840, while that of 4-D torus is 768, the overall aggregated network bandwidth of Dragonfly is much higher than that of 4-D torus. This resulting in the higher relative performance improvement of 4-D torus because of the lack of the overall network bandwidth for the baseline performance. Indeed, the absolute maximum FT performance of Dragonfly is more than twice as much as that of 4-D torus, in opposite to the lower relative performance improvement of Dragonfly.

Besides the link bandwidth, the switch delay affects a little to limited benchmarks, IS of Torus and Dragonfly, and MM and Graph500 of Dragonfly. Not only the higher link bandwidth but also the lower switch delay are crucial for higher average parallel applications performance. In this context, the swappable hash function becomes effective. high link bandwidth.

VI. Alternative ways to 100% Cache Hit Rate

There are two alternative ways to target 100% cache hit rates. We qualitatively discuss them.

A. Host-Assisted Software Cache

A simple way to fill the cache entries that will be immediately used is a prefetch. A compute node that will generate a message soon sends an empty packet to the same destination in order to fill out the cache entry at all switches on the route. When it forwards the successor message, intermediate switches will hit its cache entry at a cache. Recently, MPI v3.0 or later supports asynchronous communication and one-side communication, and we can use it as the empty packet for the cache prefetch. However, if all-to-all communications occur, it is impossible to store all packet destinations in the cache. In this context, the cache prefetch would be impractical as the number of compute nodes increases.

B. MPI-Aware Cache Refill Algorithm

Another approach is to optimize a cache refill algorithm. When a switch understands MPI communication pattern, it may improve the cache refill algorithm. Once a switch detects an all-to-all communication, it can identify a cache entry whose packet is already transferred in this all-to-all communication. The switch can eject the entry, and prefetch another entry that will be used soon in the all-to-all communication. However, we consider that this operation becomes complicated, and it is unclear whether a 100% hit rate maintains or not as the number of compute nodes increases.

VII. Conclusions

We explore an on-chip packet forwarding cache to a switch to resolve the delay and throughput problems of routing decision. Since an incoming packet avoids large-latency accessing a TCAM forwarding table if the cache hits, both throughput

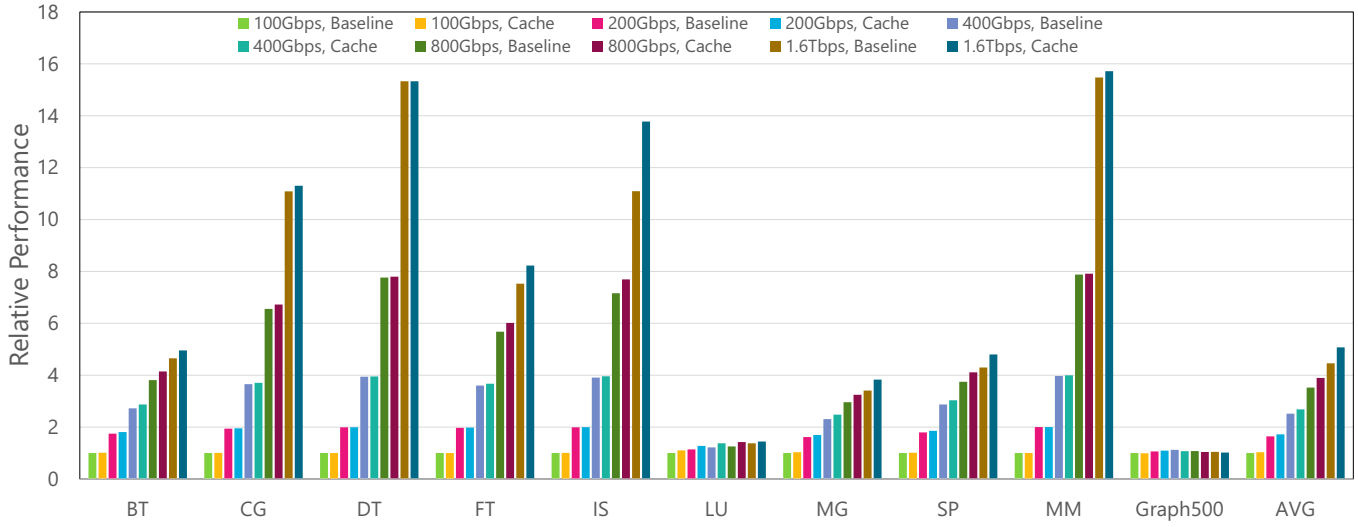


Figure 17. Relative Application Performance (4-D Torus).

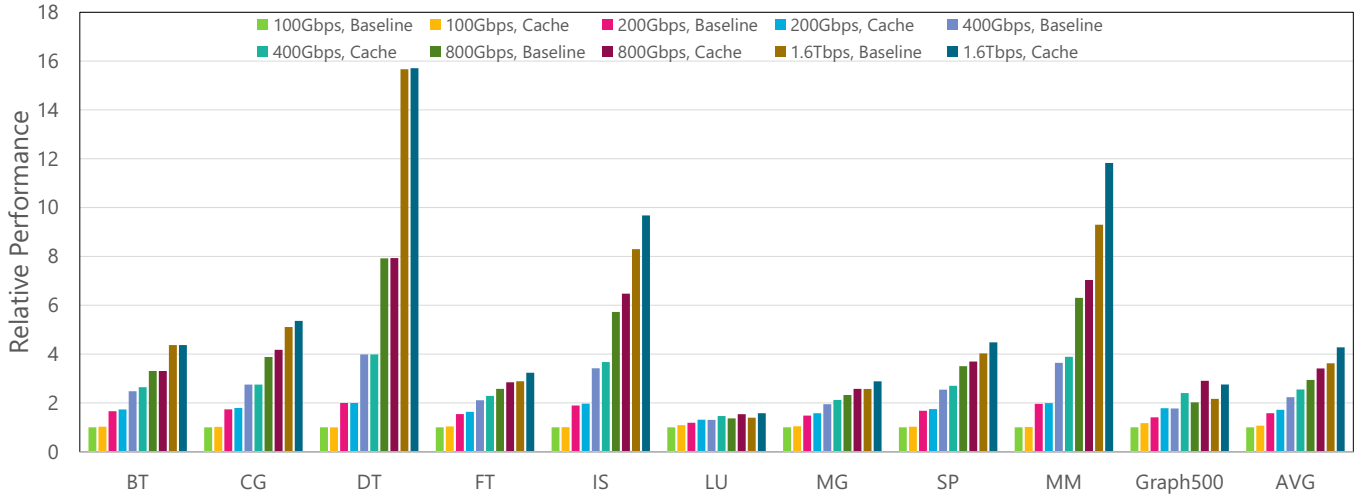


Figure 18. Relative Application Performance (Dragonfly).

and switch delay is much improved. Although we illustrate the conventional cache switch architecture, it becomes a low hit ratio close to *zero* if the network size becomes large. To achieve an almost 100% hit rate on any network size, we present a switchable hash function to index a packet forwarding table on a switch. The switchable hash function is optimized to typical network topologies, e.g., k-ary n-cubes, fat tree, and Dragonfly, thanks to custom node addressing.

The CACTI simulation results show that the reasonable packet forwarding cache supports up to 933 Gbps line rate on the above network topologies. It implicitly suggests that the packet forwarding cache enables a top-of-rack high-radix switching ASIC, e.g., 51.2 Tbps (800Gbps $\times$ 64), with the moderate chip-area overhead for incoming shortest packets.

We illustrate that parallel applications obtain the performance gain of 5.07x speed up using the cache switches since the cache switch reduces the delay of the routing computation unit.

Acknowledgements

The authors thank Dr. Yao Hu, National Institute of Informatics, Japan, for creating Figure 3 for us. This work was supported by JSPS KAKENHI 19H01106 and JST PRESTO JPMJPR19M1.

References

[1] Top 500 Supercomputer Sites. http://www.top500.org/.
[2] K. Scott Hemmert et al, "Report on Institute for Advanced Architectures and Algorithms, Interconnection Networks Workshop 2008." http://ft.ornl.gov/pubs-archive/iaa-ic-2008-workshop-report-final.pdf.
[3] Intel Tofino2, "https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html," 2021.
[4] Ethernet Alliance. https://ethernetalliance.org/.

[5] RENESAS Electronics Corp., "Network Search Engine." https://www.renesas.com/jp/ja/p roducts/memory/application-specific-memory/network-search-e ngine.html.

[6] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh, "Viking: A Multi-Spanning-Tree Ethernet Architecture for Metropolitan Area and Cluster Networks," in *Proc. of 23th Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, pp. 2283–2294, Mar. 2004.

[7] M. Koibuchi, T. Otsuka, T. Kudoh, and H. Amano, "A Switch-Tagged Routing Methodology for PC Clusters with VLAN Ethern et," *IEEE Transaction on Parallel and Distributed Systems*, vol. 22, pp. 217–230, Feb. 2011.

[8] Rockley Photonics demos co-packaged optics OptoASIC Switch system with multiple partners, "https://www.lightwaveonline.com/optical-tech/components/article/14170143/rockley-photonics-demos-copackaged-optics-optoasic-switch-system-with-multiple-partners," 2021.

[9] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *Passive and Active Network Measurement, 10th International Conference, PAM*, vol. 5448 of *Lecture Notes in Computer Science*, pp. 3–12, 2009.

[10] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging zipf's law for traffic offloading," *Comput. Commun. Rev.*, vol. 42, no. 1, pp. 16–22, 2012.

[11] H. Yamaki, H. Nishi, S. Miwa, and H. Honda, "Data prediction for response flows in packet processing cache," in *Proc. of the 55th Annual Design Automation Conference (DAC 2018)*, no. 110, pp. 1–6, 2018.

[12] N. P. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proceedings of the Symposium on SDN Research, SOSR* (B. Godfrey and M. Casado, eds.), p. 6, 2016.

[13] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie, "Towards tcam-based scalable virtual routers," in *Conference on emerging Networking Experiments and Technologies, CoNEXT'12* (C. Barakat, R. Teixeira, K. K. Ramakrishnan, and P. Thiran, eds.), pp. 73–84, 2012.

[14] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[15] W. D. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[16] D. K. Dror G. Feitelsona, Dan Tsafrirb, "Experience with using the Parallel Workloads Archive," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2967–2982, October 2014.

[17] Nangate Inc., "45nm Open Cell Library." http://www.nangate.com/openlibrary/, 2008.

[18] SimGrid: Simulation of Distributed Computer Systems. https://simgrid.org/.

[19] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Sut er, "Versatile, Scalable, and Accurate Simulation of Distributed Applica tions and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[20] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performane," in *NAS Technical Report NAS-99-011*, Oct. 1999.

[21] Graph 500 — large-scale benchmarks. http://www.graph500.org/.