The Application of Routing Cache for High-Bandwidth Low-Latency Switch on Interconnection Networks

by

Shoichi Hirasawa

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI
March 2022

A dissertation submitted to Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies, SOKENDAI,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

1. Prof. Kento AIDA	National Institute of Informatics, SOKENDAI
2. Assoc.Prof. Michihiro KOIBUCHI	National Institute of Informatics, SOKENDAI
3. Prof. Masahiro GOSHIMA	National Institute of Informatics, SOKENDAI
4. Prof. Atsuko TAKEFUSA	National Institute of Informatics, SOKENDAI
5. Emer.Prof. Kei HIRAKI	Preferred Networks, Inc., The University of Tokyo

"Standing on the shoulders of giants."

— John of Salisbury, Isaac Newton, and Google Scholar.

Acknowledgements

I would first want to represent my thankfulness to Professor Aida. He has always given me many helpful advises and useful suggestions. This work will not be completed without his thoughtful help.

In addition, I would like to express my profound gratitude to Professor Koibuchi. He guided me about this work, and I have learned much from him how to write papers. He always reads my draft papers carefully and gives me the most valuable suggestions. I also appreciate his financial support on our laboratory to keep comfortable research environment.

I want to appreciate Prof. Goshima, Prof. Takefusa, and Prof. Fukuda for their constructive comment to improve my work. They kindly read my dissertation and gave me many precious suggestions.

I would like to deeply thank Professor Hiraki. He always encourages me whenever I went through difficulty in research and all other activities.

I would like to deeply thank Professor Yamaki for his commitments, suggestions, and comments on proceeding this work.

I will thank all members of Koibuchi laboratory for their useful advice and encouragement. I can always argue with them about my problems and they are always very willing to present much of their acquaintance and precious knowledge.

And finally, I give my special thanks and affection to my wife and our children. Even I lost confidence in my work and, sometimes even worse, confidence of myself, they definitely trust me and give me willingness and strong motivation.

THE GRADUATE UNIVERSITY FOR ADVANCED STUDIES, SOKENDAI

Abstract

School of Multidisciplinary Sciences
Department of Informatics

Doctor of Philosophy

The Application of Routing Cache for High-Bandwidth Low-Latency Switch on Interconnection Networks

by Shoichi Hirasawa

Parallel applications become sensitive to communication latencies and bandwidth of interconnection networks between compute nodes on parallel computers. Interconnection networks have been studied for parallel computers, including supercomputers and high-end datacenters. Switch delay dominates communication latencies in interconnection networks, especially for short messages because switch delays are massive compared 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. A main problem of the packet forwarding processing is the significant operation latency to the CAM at a switch. Reducing the CAM access latency 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 be able to forward incoming packets at the maximum line rate. It is also difficult to provide the proportional packet forwarding rate to a high line rate on a future switch even for long packets. The key design to resolve the problem of the packet forwarding performance is a packet forwarding cache architecture explored in this dissertation. More precisely, "address patterns" of interconnection networks should be found, and the packet forwarding cache architecture should be optimized for enjoying the address patterns.

To resolve the latency and throughput problems, an on-chip packet forwarding cache to a switch is explored. An incoming packet avoids large-latency of accessing a CAM forwarding table if the cache hits. Firstly, the cache for up to 2K-node jobs is optimized because the sizes of a large number of workloads are less than 2K compute nodes. The conventional cache design achieves an almost 100% hit rate (no capacity miss nor conflict miss) for packets generated in up to 2K-node jobs on arbitrary network topologies, which affect the access pattern of a switch. 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. Zero-load communication latency with the packet forwarding cache in a large scale interconnection network is evaluated. From the evaluation results, the reduction percentage of zero-load communication latency gradually decreases from 19% to 13% with the effects of capacity misses of the packet forwarding cache when used in a 9K computation node large scale interconnection network. Additionally, with additional entries in the packet forwarding cache, the reduction percentage of zero-load communication latency gradually increases from 9% to 19%. The adoption of the packet forwarding cache clearly decrease the communication latency of interconnection network and increase both the line rate and the performances of parallel applications. Consequently, the packet forwarding cache is strongly recommended to be adopted in HPC switches. However, larger jobs make the cache hit rate almost "zero" on any network topologies, and the cache effect becomes almost "zero."

Secondly, a switchable node reduction function to refer to a packet forwarding table on a switch is presented for 100% hit rate on larger jobs. 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. A general node reduction function, which obtains two addresses and their indices then returns a cache tag, is defined to achieve the path regularity. The switchable node reduction function is then optimized to typical network topologies, i.e., k-ary n-cubes, fat trees, and Dragonfly. 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. In addition, they 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.

Through this dissertation, it is concluded that a commodity switch should have a packet forwarding cache with switchable node reduction functions. The packet forwarding cache is efficient for forwarding a large number of shortest packets, and the switchable node reduction function is necessary for large scale parallel computers.

Contents

List of Figures

List of Tables

To my family

Chapter 1

Introduction

1.1 Motivation

Interconnection networks have been studied for parallel computers, including supercomputers and high-end datacenters. They have four key architectural components, network topology, routing, switching, and arbitration. The main concern of this dissertation is the network topology and routing.

Unlike custom interconnection networks, e.g. BlueGene/Q [6] and Tofu Interconnect D (TofuD) [7] for Fugaku supercomputer [8], commodity interconnection networks, e.g., Ethernet [2] and InfiniBand [3], are a mainstream of supercomputer's communication platform. As of June 2021, more than 80% of Top500 [1] supercomputers use them, as shown in Figure 1.1. A commodity interconnection network usually supports arbitrary network topologies and arbitrary routings by updating a packet forwarding table at its network switch.

At a conventional commodity switch, routing decision is based on CAM [9] (Content Addressable Memory)-based table lookup, though there are two routing implementations, distributed table lookup and source routing. Source routing packs the routing information to the destination into a packet header, thus requiring no routing tables at an intermediate switch. Unlike commodity local area networks, the next-generation High Performance Computing (HPC) systems will take aggressive network feature, such as congestion control and adaptive routing, which dynamically selects a route from

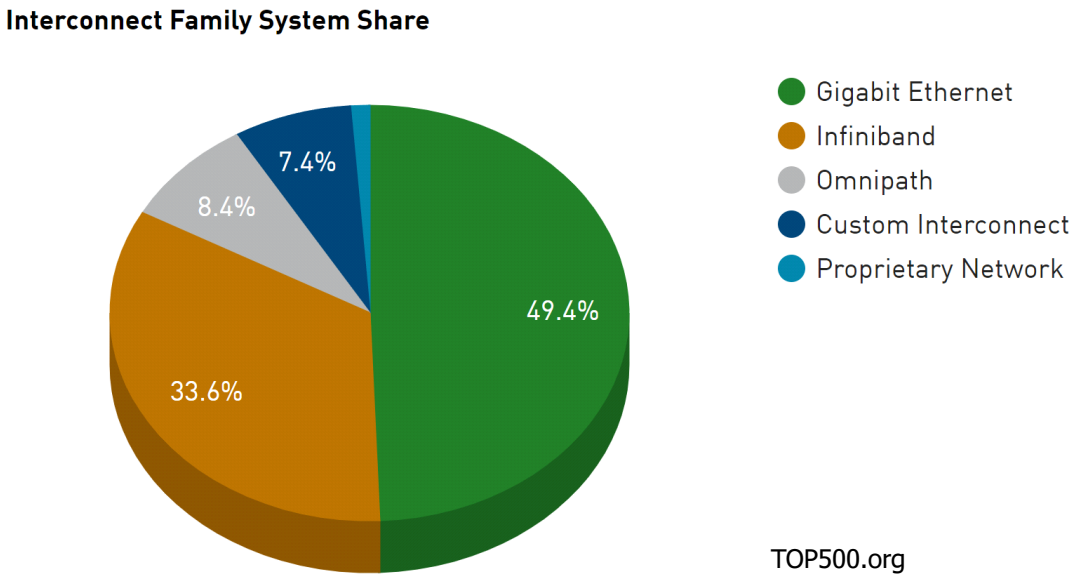


FIGURE 1.1: Interconnect Family System Share of TOP 500 Release June 2021 [1].

alternative routes at an intermediate switch so as to avoid congestion points in an interconnection network. Source routing cannot support adaptive routing. This dissertation considers the case for the table lookup for routing at a switch.

A main problem of the packet forwarding processing is the significant operation latency to the CAM at a switch. The next-generation HPC systems are to achieve low end-to-end latencies, e.g., under 1 μs across an exascale system [10]. 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 [11] 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 400 Gbps $\times$ 64 ports in a cutting-edge switch [12]. Link bandwidth continues to increase to 1.6 Tbps in the 2020s, as illustrated in Ethernet Roadmap 2020 [2], 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 [13] in 2009, while four nanoseconds for the successor R8A20611BG in 2021 [14].

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 (= 1/4 ns × 84 Bytes × 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 (=1/4 ns × 1,518 B ×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 (=1/4 ns × 1 KiB × 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. This dissertation challenges to mitigate this problem.

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 [12]. Of course, this parallelism approach will be unrealistic due to the vast cost as the line rate increases.

1.2 Challenges

To improve the operation latency to compute routes, i.e., the packet forwarding rate, an on-chip packet forwarding cache to a switch is explored in this dissertation. 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., 2 ns) that do not provide a high line rate, i.e., higher than 336 Gbps (= 1/2 ns × 84 Bytes × 8). The L1 cache size is strictly limited by the chip area. This limitation strongly affect the cache hit rate for traffic generated in parallel applications.

Access locality is a key factor to make the cache hit rate high. However, parallel applications sometimes generate traffic with less access locality. For example, alltoall collective communication makes access pattern uniform. Thus, two conditions are taken for making strong access locality of a cache as follows. (1) up to 2K job sizes, and (2) supporting only typical network topologies with their custom routing. In this dissertation, a packet forwarding cache architecture is exploited for improving the link bandwidth and routing computation latency of a network switch for the two above cases.

1.3 Contributions

Firstly, a packet forwarding cache architecture on a network switch is illustrated. 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). However, the solution of the packet forwarding cache cannot work well when the network size is large.

Secondly, a switchable node reduction function for the packet forwarding cache architecture is demonstrated for elephant-nose large jobs on some predefined network topologies. Some typical configurations of the functions are stated. It requires taking a custom packet destination addressing on the predefined network topologies.

Table 1.1 highlights the contributions of this dissertation.

TABLE 1.1: Contributions for High-Throughput, Low-Latency Packet Forwarding of a Network Switch.

Section	Item	Detail
Chapter 4	Assumption	Arbitrary network topologies.
	Limitation	Job size is up to 2K compute nodes.
	Solution	The four-way set associative cache.
	Efficiency	99.9% hit rate. 1.64 Tbps link, 4.39x application speedup.
Chapter 5	Assumption	Arbitrary number of compute nodes.
	Limitation	Predefined topologies.
	Solution	The switchable node reduction function.
	Efficiency	933 Gbps link. 5.07x application speedup.

1.4 Organization

The remainder of the dissertation is organized as follows. Chapter 2 describes background information of this dissertation. Chapter 3 states the problem mitigated in this dissertation. Chapter 4 describes the switch organization using the packet forwarding cache. Chapter 5 describes the switchable node reduction function on the switch with the packet forwarding cache. Chapter 6 discusses alternative ways to improve the hit rate of the packet forwarding cache. Chapter 7 concludes with a summary of the findings and perspectives on future work.

Figure 1.2 illustrates the outline of this dissertation.

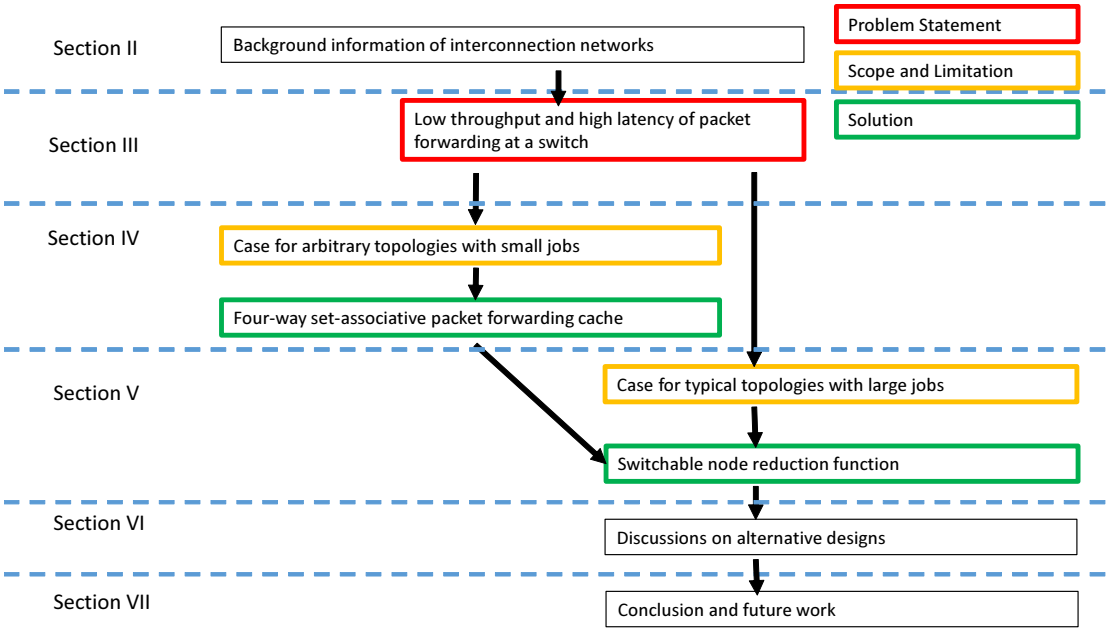


FIGURE 1.2: Outline of the Dissertation.

Chapter 2

Background Information

2.1 Parallel Computers

Energy consumption and heat problem limits the performance of single CPU cores and the number of cores in a die. The performance of shared-memory computers suffer from the long latency of a shared main memory in an Uniform Memory Access (UMA) system, or the complexity of cache coherence in a cache-coherent Nonuniform Memory Access—(ccNUMA) system. Consequently, distributed memory computers with message passing architectures are widespread in large scale parallel computers.

Parallel computers, including supercomputers, continues to increase the number of processing cores. As for June 2021, Fugaku, which is the first rank of Top500 [1], takes 7,630,848 processor cores. ABCI [15], Piz Daint [16], and Titan [17] have 391,680, 387,872, and 560,640 processor cores, respectively.

From the interconnection-network point of view, the interest of this dissertation is the number of endpoints and their types on parallel computers. Each compute node usually consists of multiple processor sockets, and each processor socket has tens of processor cores. Piz Daint supercomputer connects 5,320 XC50 [18] compute nodes. Titan supercomputer has 200 racks to store 18,688 compute nodes. Some parallel computers take not only commodity processors but also accelerators, e.g., General Purpose Graphics Processing Units (GPGPU) [19], for increasing the system performance. For example, Cori supercomputer [20] consists of 2,388 Haswell CPU compute nodes and 9,688

KNL [21] compute nodes. In ABCI, each compute node also has commodity processors and GPUs, and ABCI has 1,088 compute nodes.

2.2 Interconnection Network

An interconnection network plays a key role to connect all of compute nodes. They have four key architectural components, network topology, routing, switching, and arbitration, as presented in a textbook [22]. Network topology determines possible routes for transmitting packets. Interconnection networks usually share paths among different pairs of compute nodes. Once the network topology is determined, the next concern is routing. Routing determines allowable possible paths for transmitting packets. Arbitration determines the available time of the path for packets when they can be transmitted. Switching determines the allocation of packets to paths.

In this dissertation, the concern is topology and routing. Arbitration and switching are well explained in textbooks [23, 24].

2.2.1 Network Topology

Interconnection networks are surveyed for top 30 supercomputers, as of Nov. 2019 [25]. Table 2.1 lists up their taxonomy. Although there are a large number of network topologies, including the results of Graph Golf competition [26], only a small number of typical network topologies, e.g. torus, fat tree [27], and Dragonfly [28], are used. In the figure, the fat tree topology takes two layers, and switches of different tiers have different numbers of ports. Tier 1 means a root switch, as known as director switch. Tier 2 means a leaf switch, or a Top of Rack (ToR) [29], which is usually located on the top of the computer racks. A number in parentheses is their ranks in Top500 Ranking [1]. The node bandwidth is the aggregate link bandwidth of compute node. For example, the first ranked supercomputer has two 100 Gbps links at a compute node, resulting in a 200 Gbps node bandwidth.

The link bandwidth varied from 16 Gbps to 100 Gbps on different supercomputers. The link bandwidth depends on the year of the system deployment. As illustrated in the Ethernet and InfiniBand roadmaps in Figure 2.1 and Figure 2.2, the link bandwidth increases year by year, almost 10 times per 10 years. In the figure, “Others” includes the case where a network topology is not disclosed.

The network topologies on supercomputers can be classified into direct and indirect. In direct network topologies, every switch connects directly to a number of compute

TABLE 2.1: Network Topologies of Top 30 Supercomputers as of Nov. 2019.

Topology	Num. of Switch Ports	Link/Node BW(Gbps)	Rankings
Fat Tree (indirect)	36/648[1,2,10] 32/576[4] 40/800[5] 36/648[8] 48/768[14,15,19,23,30] 648[20]	100/200[1,2,8,11] 112/112[4] 200/100[5] 100/100[14,18,19,30] 100/800[20,26,29] 100/400[23]	1,2,4,5,8-11, 14,15,17-20, 23,24,26,29,30 (19 Machines)
Dragonfly (direct)	48	37.5-42/84	6,7,13,27,28 (5 Machines)
5-D Torus (direct)	10	16/Integrated	12,22 (2 Machines)
Others (inc. unknown)	-	-	3,16,21,25 (4 Machines)

nodes and other switches. Direct topologies are k-ary n-cubes and Dragonflies in top 30 supercomputers. All these network topologies have the same degree.

k-ary n-cubes include tori and meshes. Their difference is the existence of wraparound channels. We illustrate 4-ary 2-torus and 4-ary 2-mesh in Figure 2.3.

The dragonfly is a hierarchical network with three levels: switch, group, and system [30]. At the bottom level, each switch has local p endpoints, $a-1$ links to other switches in the same group,and h global links to switches in other groups [30]. Thus, the switch degree become $p+a+h-1$. The main idea of Dragonfly is that a group, consisting of a switches, behaves as a single virtual switches with a degree of $a(p+h)$, which can be considered as a high-degree network. It is a rack-conscious topology that distinguishes the intra-rack layer and the inter-rack layer, which can be seen as an instance of the Dragonfly.

The original Dragonfly is described as "three-layered," including compute nodes, but in this dissertation we ignore compute nodes and consider only intra- and inter-rack links. Figure 2.4 is an example of Dragonfly network topology, in which inter-group network is a ring, while intra-group network is a fully connected network topology. Inter-group links are marked as red color, while intra-group links are marked as black color. The other layout-conscious network topologies provided a better floorplan for reducing the network latency or network costs [31–34].

In indirect network topologies, some switches are connected only to other switches. Famous indirect network topologies are fat trees, Clos network [35] and multi-stage interconnection networks [36] (MINs) such as the Omega [37] and Butterfly [38] networks.

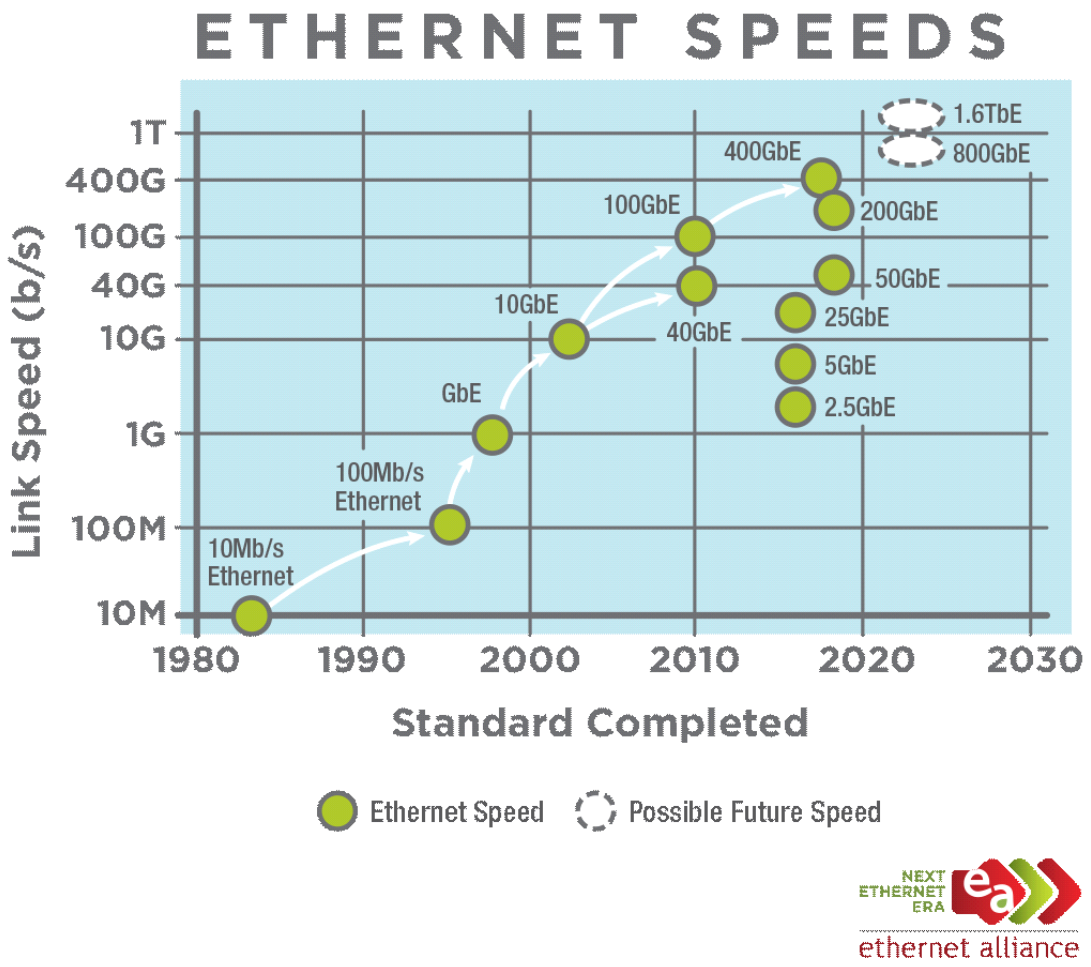


FIGURE 2.1: The 2018 Ethernet Roadmap [2].

MINs have uniform access latency and they use different schemes by which link end points are "shuffled" deterministically at each stage, so that re-arrangeable or non-blocking data transfers are possible. Fat tree consists of multiple trees, and recent supercomputers follow the two-layer structure, director switch (some hundreds of ports) and Top-of-Rack (ToR) switches (some dozens of ports), as shown in Figure 2.5.

ToR switches connect both compute nodes and neighboring switches, while the director switch only connect neighboring switches. In this context, fat tree is an indirect interconnection network. There are some variations of the two-level fat trees. A simplest approach is to have a single link[1] between compute node and ToR switch. Another approach takes multiple links from a compute node to different ToR switches.

[1] If link aggregation is taken, it should be counted as one.

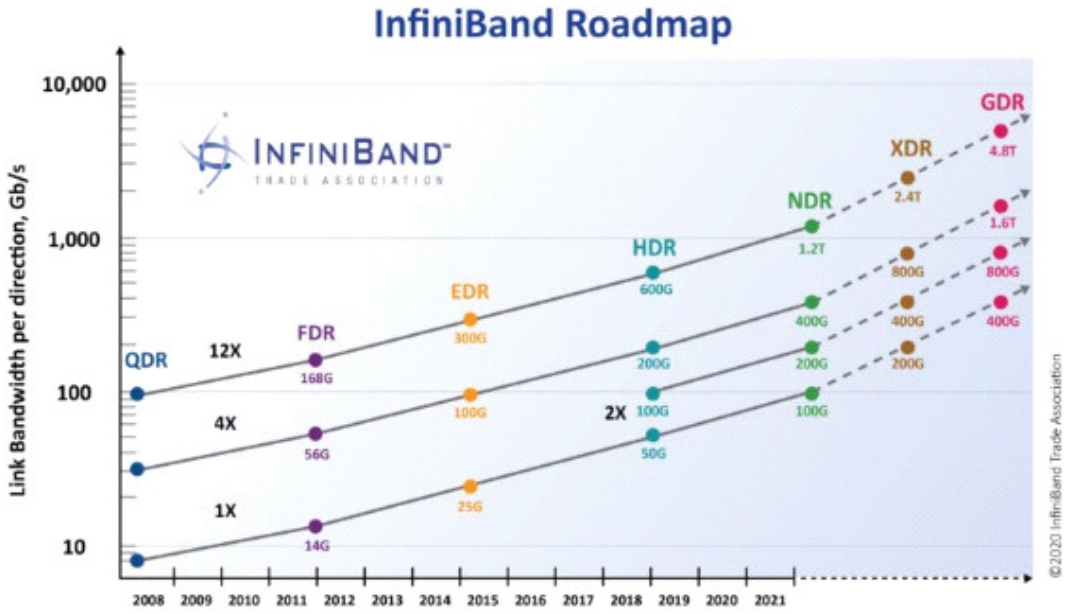


FIGURE 2.2: IBTA's InfiniBand™ Roadmap [3].

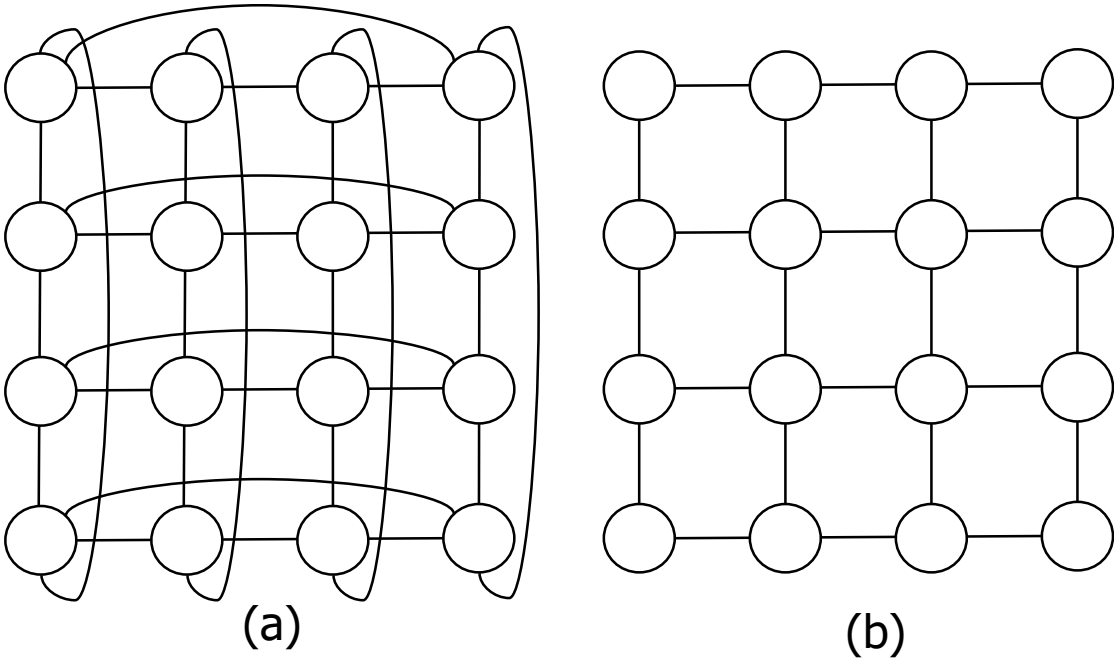


FIGURE 2.3: Example of k-Ary n-Cubes (a) 4-Ary 2-Torus and (b) 4-Ary 2-Mesh.

2.2.2 Routing

Interconnection networks require the routing to resolve deadlocks of packets because they are "lossless" using virtual-cutthrough or wormhole switching. Custom deadlock-free routing algorithms have been proposed and used in supercomputers for the above typical network topologies, e.g., dimension-order routing and Duato's protocol for k-ary

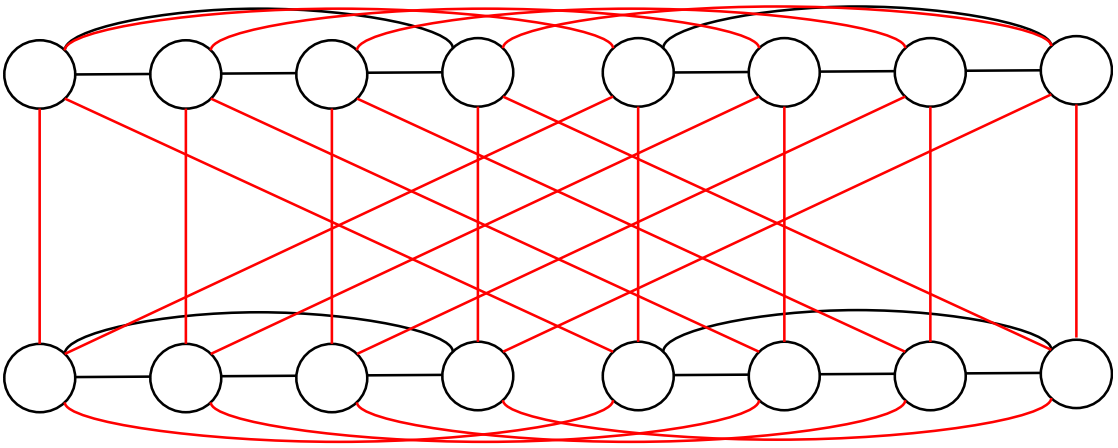


FIGURE 2.4: Example of Dragonfly.

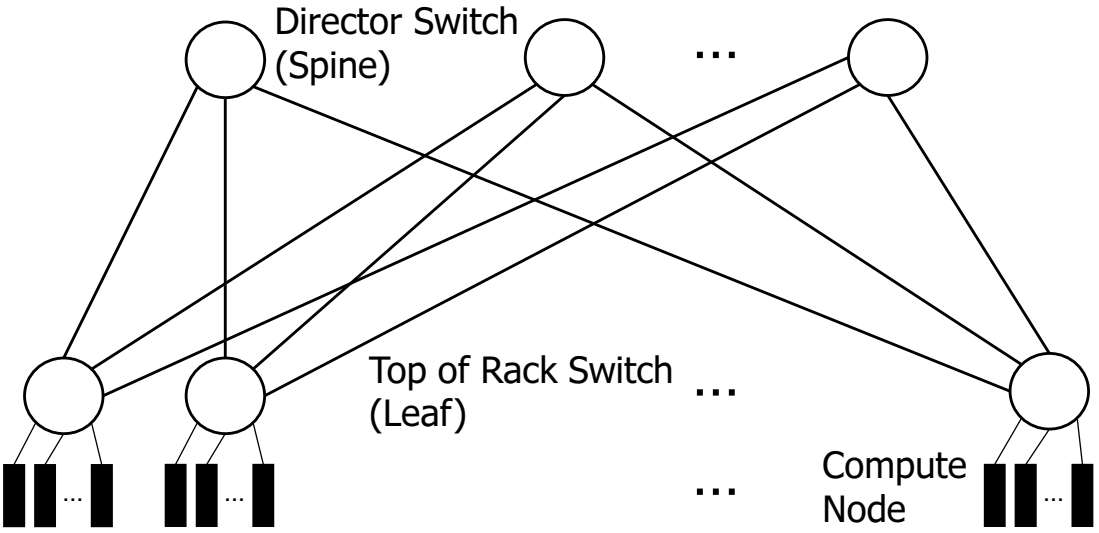


FIGURE 2.5: Example of a Two-Level Fat Tree.

n-cubes [23]. Routing on these network topologies includes a strong regularity that helps design a routing algorithm.

Besides the above custom routing to specific network topologies, topology-agnostic deadlock-free routing algorithms are well studied [39, 40], and they can be applied to any network topologies.

Routing algorithms are classified into deterministic and adaptive methods. The former determines a single path for a given pair of source and destination nodes. The latter allows a packet to dynamically select a path from alternative paths for avoiding the congestion at an intermediate switch.

There are two approaches to avoid deadlocks in routing. A simple approach breaks cyclic

channel dependency of paths [41]. A famous example uses a spanning tree [42] embedded in the network topology. All deterministic and many adaptive routing algorithms follow this approach. Up*/down* routing is based on the assignment of direction, up or down, to network channels using a spanning tree [43]. A legal path must traverse *zero* or more channels in the up direction followed by *zero* or more channels in the down direction [43]. Since no cyclic channel dependencies are formed among paths with the above restriction, deadlock freedom is guaranteed. Since there are multiple legal paths to a destination, up*/down* routing is an adaptive routing algorithm. Sophisticated deadlock-free routings are designed using virtual channels for taking minimal paths [39].

Another approach to avoid deadlocks prepares both escape and adaptive paths over the network. Some adaptive routing algorithms follow this approach [23, 44]. It provides deadlock freedom of packets while cyclic channel dependencies are allowed. They take different virtual channels for separating network resources. Along virtual channels for adaptive paths (adaptive virtual channel), each packet can take any route. When it is blocked at an intermediate switch, it has to select a virtual channel along the escape path (escape virtual channel). In an escape virtual channel, the route has to be deadlock freedom among escape paths. Once a packet follows an escape virtual channel, it should be transferred only with escape virtual channels [44]. arbitrary network topologies. More recently, through the detailed analysis of the condition of deadlocks, deadlock-free routings with the higher adaptivity are discovered [45].

Commercial commodity interconnection networks, e.g., InfiniBand and Ethernet support deterministic routing. However, adaptive routing dynamically selects a route from alternative routes at intermediate switches, depending on the network congestion. As a result, adaptive routing will be one of the key features for the next-generation high-bandwidth low-latency interconnection networks for supercomputers.

2.2.3 Switch Microarchitecture

The baseline switch microarchitecture is described in this section. A four-stage router quoted from [46] is assumed as a baseline switch in this paper as shown in Figure 2.6. The blue arrows in the figure represents data movement of a packet. The black arrows in the figure shows controls to calculate the path of a packet in the switch.

Each packet arrives at an Input Unit. The output port of the packet is determined by Routing Computation. VC Allocator allocates an output virtual channel for the packet. Switch Allocator allocates a time slot of Crossbar to forward the packet.

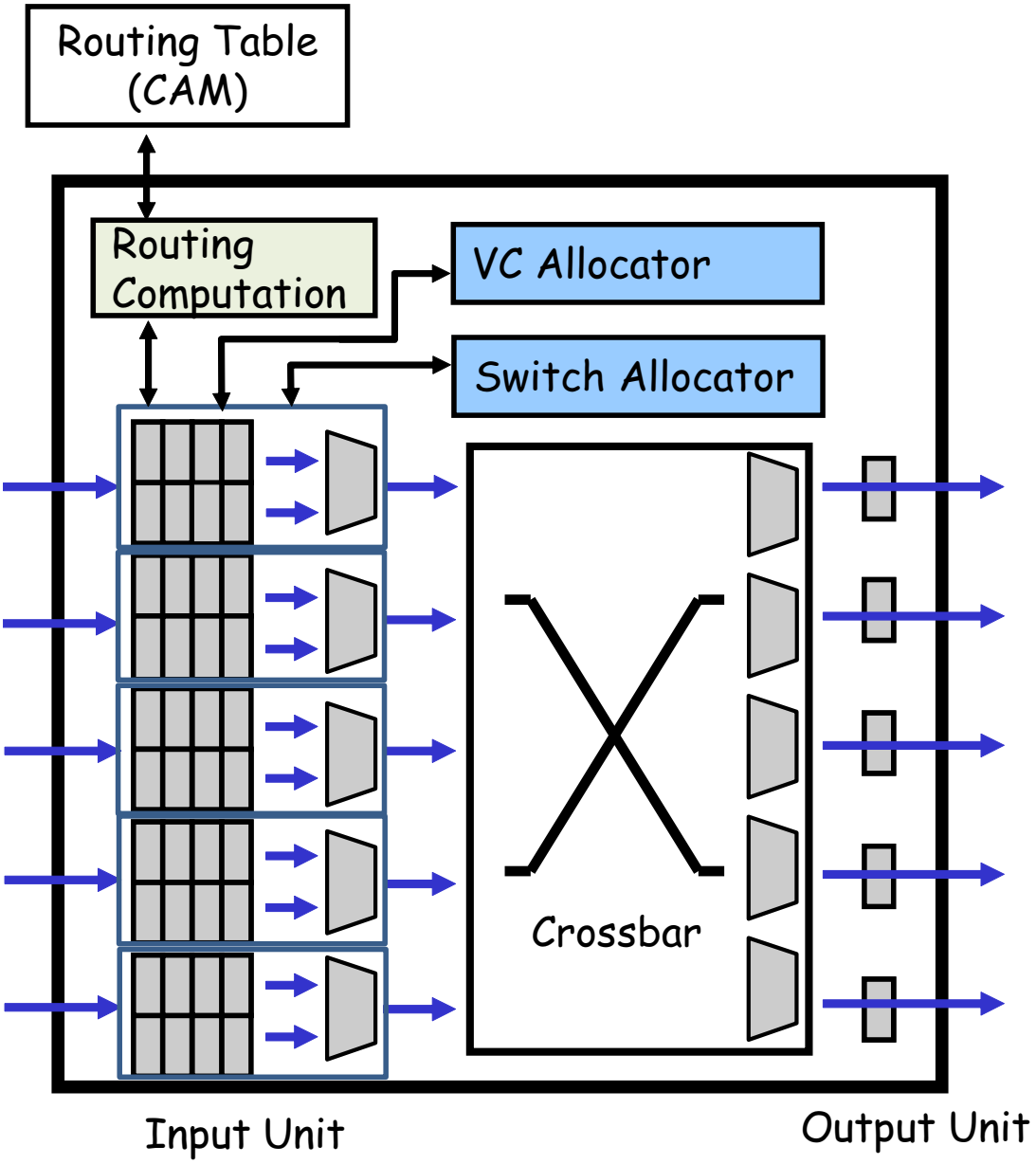


FIGURE 2.6: Block Diagram of a Traditional Switch.

Similarities with vector processors, the key design is pipelining the switch microarchitecture. In the switch, a header flit is transferred through four pipeline stages that consist of a routing computation (RC) stage, a virtual channel allocation (VA) stage for output channels, a switch allocation (SA) stage for allocating the time-slot of the crossbar switch to the output channel, and a switch traversal (ST) stage for transferring flits through the crossbar. An example of a four-flit packet transfer is illustrated in Figure 2.7.

Indeed, each of the four operations is executed with multiple clock cycles. The access latencies and the cycle numbers of each pipeline stage are not publicly available in many

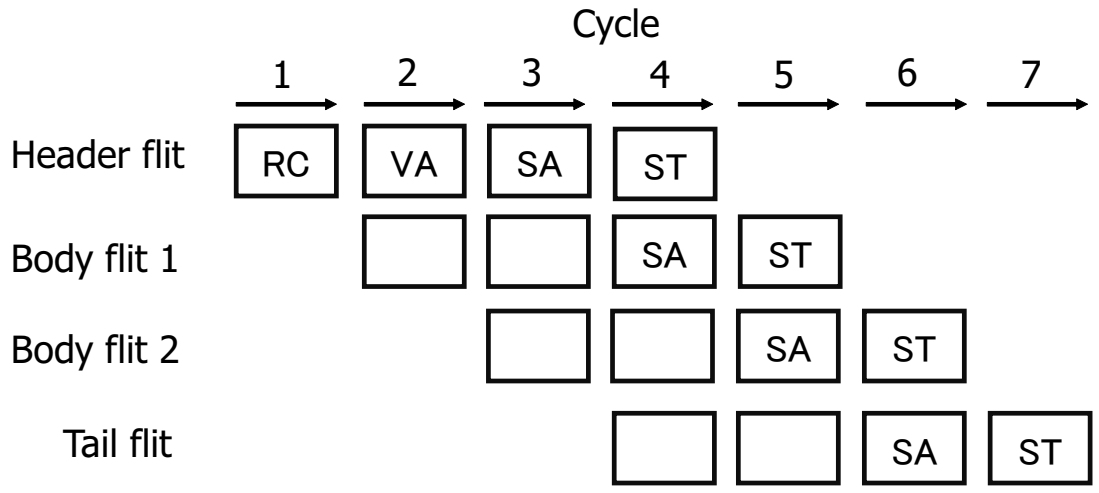


FIGURE 2.7: Four-stage Pipeline Processing of a Packet.

commercial switches, with some exceptions, e.g. IBM BlueGene/Q router is 40 nano seconds(500MHz, 20 cycles) [47], Cray YARC router is 31.25 nano seconds(800MHz, 25 cycles) [48], Fujitsu 10GbE switch is 450 nano seconds(312.5MHz, 140 cycles) [49], RHiNET-2/SW is 160 nano seconds(125MHz, 20 cycles), and RHiNET-3/SW is 240 nano seconds(100MHz, 24 cycles) [50]. Although the parallelism of control dependency on the pipeline structure can be logically relaxed by look-ahead routing and speculation [24], actual switch products still have over 100 nano seconds delay.

In this dissertation, the pipeline stages other than the RC stage are not improved. Hence, they are not treated hereafter.

2.2.4 Optical Network

WDM(Wavelength Division Multiplexing) is commonly used in optical fiber communications. In WDM, each data carrier is represented as a wavelength, whereas multiple wavelengths are multiplexed to a single fiber for transmission, as shown in Figure 2.8. Consequently, multiplication of capacity of data is transferred with a single fiber.

DWDM(Dense Wavelength Division Multiplexing) is a WDM technique using denser wavelengths, whose differences are around 0.8nm, than the wavelengths of CWDM(Coarse Wavelength Division Multiplexing), whose differences are around 20nm. Because of the small differences of wavelengths, DWDM can transmit more data than CWDM, resulting in a higher bandwidth.

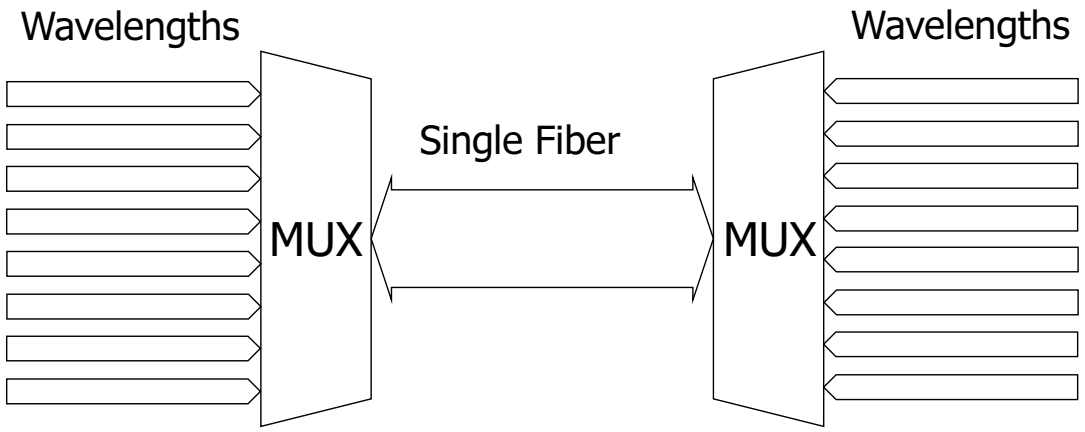


FIGURE 2.8: WDM(Wavelength Division Multiplexing) optical transmission.

2.3 Network Cache Architecture

In this dissertation, a cache structure for forwarding a packet at a switch in interconnection networks is used. This section introduces the prior studies on network cache architecture. Cache architecture to forwarding packets has been discussed mainly for Internet routers with TCAM. The prior studies concern either high hit rate or longest prefix matching.

2.3.1 DRAM Routing Tables

In the 1990s, the routing table is stored in DRAMs whose access latency becomes some hundreds of nano seconds [51], because a series of table entry comparison is necessary to find a matched entry from a DRAM table. CPU Caches [52] and SRAMs [53] are used to improve the lookup performance.

Figure 2.9 shows an example of a search from a DRAM routing table. In the Figure, "10001" is searched from the DRAM table. First, the first line of the DRAM table is compared to "10001," resulting in a no-match. Then, the second line of the DRAM table is compared to "10001," resulting in a match. As a result, the second port is returned as the result of the DRAM table lookup.

Figure 2.10 shows another example of DRAM table lookups. In the Figure, "00010" is searched from the table. As the entry can only be found in the fourth line of the table, four comparisons, which needs four DRAM reads, are needed to return the matched result. As a result, this search takes longer time than that of the case in Figure 2.9.

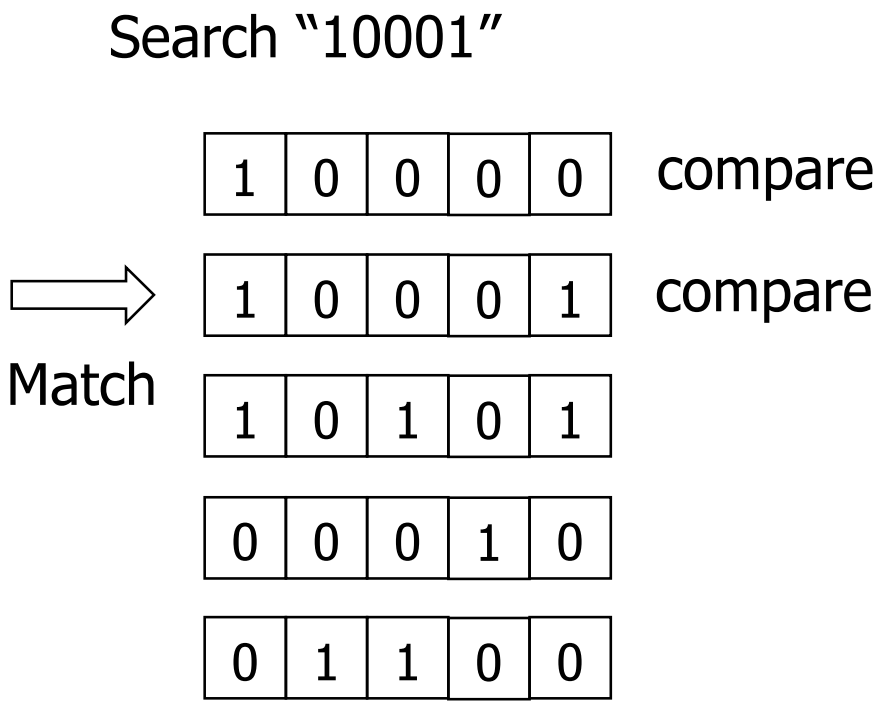


FIGURE 2.9: DRAM Table Matching; the Second Row Matches.

2.3.2 Content Address Memory (CAM)

Content Address Memory (CAM) is widely used in switch routing. Figure 2.11 shows two search examples to a CAM. CAM is a memory table, which contains multiple entries with a static length bits. When a search comes to a CAM, it returns a row number if the entry in the row matches. Exact match is checked in a binary CAM.

Ternary CAM (TCAM) is a CAM that allows the X bit (don't care bit) in its table. Figure 2.12 shows examples to search a TCAM. In the Figure, if a bit patter "10001" is searched in the TCAM, the first row is a hit because the last bit of the first row is an X bit, which matches "1" of the last bit of "10001". This search will not match to the first row of the CAM in Figure 2.11, because the CAM requires a strict match and the last bit of the first row in the CAM is "0".

Switches of specially implemented interconnection network, e.g. Anton-2 or IBM BlueGene/Q, can forward packets with special hardware synthesis. They achieve low switching latencies, e.g. 40 nano seconds per hop for Anton-2.

However, the adaptation of these specially implemented switches are limited on HPC interconnection networks. CAM is suitable to implement routing tables of switches which supports any network topologies, e.g. InfiniBand which is widely used in HPC. The entry size of the routing table is huge because it needs to accommodate the whole

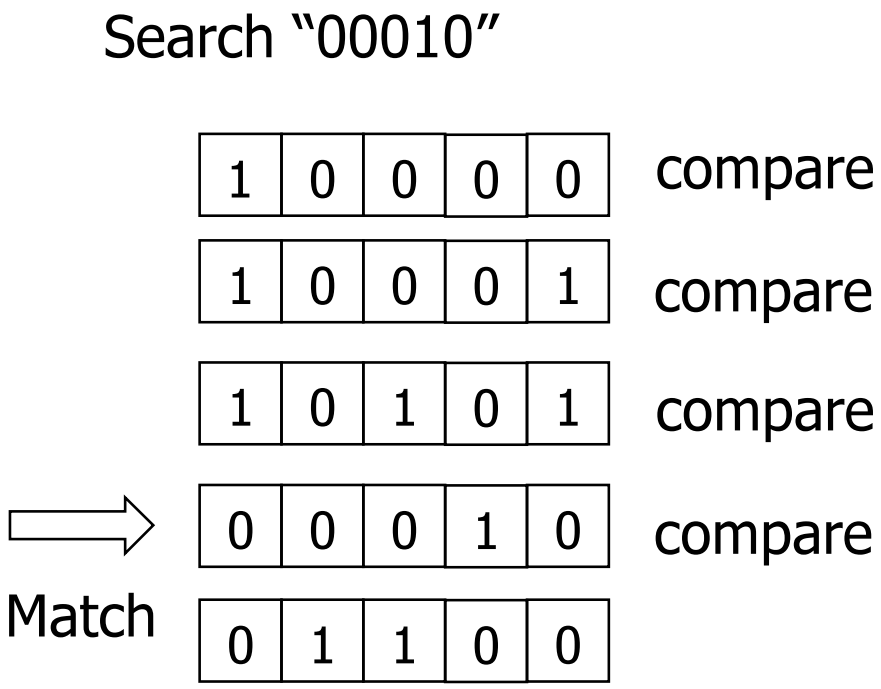


FIGURE 2.10: DRAM Table Matching; the Fourth Row Matches.

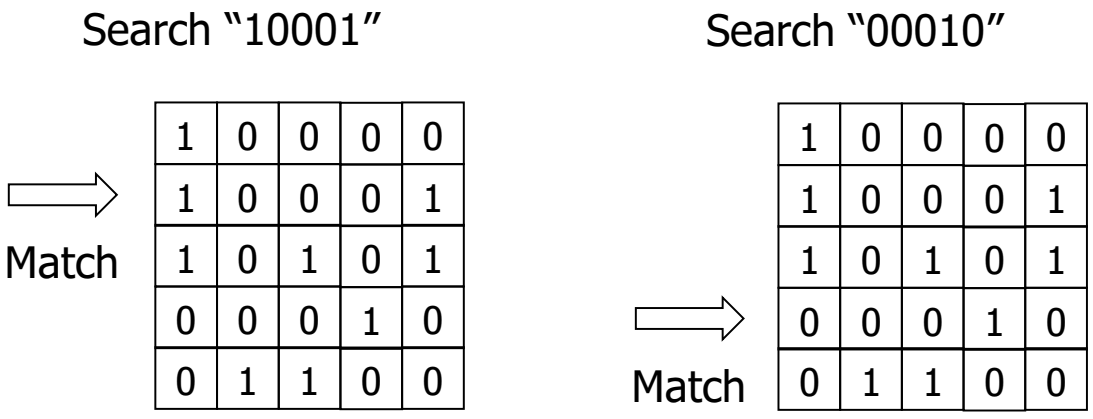


FIGURE 2.11: Binary CAM Searches.

network addresses, 16 bit local ID in InfiniBand. Consequently, CAM and the switch is implemented in different chips.

The access latency of CAM chips used in routers or switches has not been reduced for a last decade. The access latency of Renesas Electronics R8A20410BG, which is shipped in 2009, is 6 nano seconds, while the latest R8A20611BG is only improved to 4 nano seconds. The access latency from the switch chip to the off CAM chip is around 25 nano seconds, including the latency of CAM chip itself and the communication latency between the switch chip and the CAM chip, including SerDes conversions. This is about

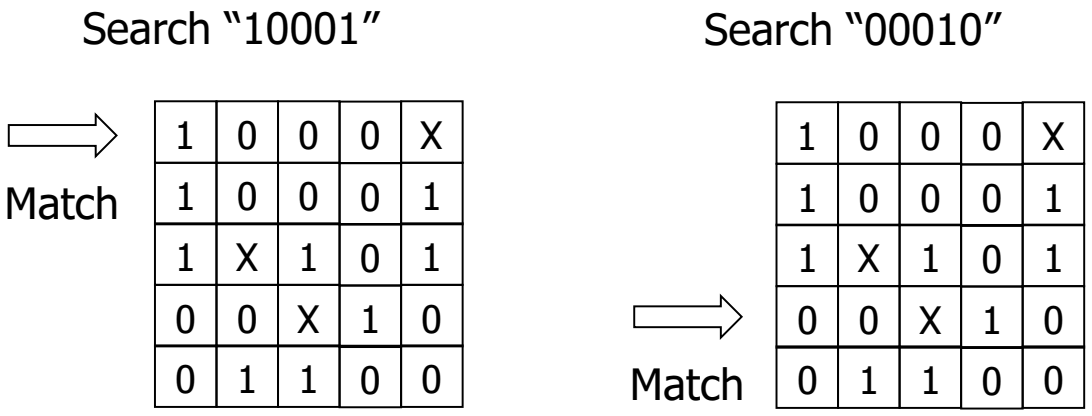


FIGURE 2.12: Ternary CAM Searches.

the same to the access latency of off-chip L3 cache chips, around 20 nano seconds [54], while the access latency of the L3 cache chips itself is only 3 nano seconds.

2.3.3 High-Throughput Cache

The fast-path technology bypasses the original datapath, and it uses cache whose access latency is some nano seconds to improve the router latency. The memory structure is similar to that in a traditional processor. Since TCAM and SRAM are immensely expensive and power-hungry than DRAM [55–58], 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 [59]. A router-specific cache replacement algorithm is proposed called traffic-aware flow offloading (TFO) [60]. 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 [60].

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 1. In this context, inserting an on-chip SRAM cache is investigated in [61]. 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.

2.3.4 Software-Defined Network

Software-defined networks [62] would require a larger number of rule tables than a number of table entries in modern TCAMs [63, 64]. 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, missrouting 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 my knowledge, there are no prior work on routing cache on parallel computer systems. The impact on the performance of parallel applications should be investigated.

2.3.5 Routing Cache

A routing cache [65] is proposed to improve the packet forwarding throughput of electric Internet routers. The access latency to the CAM is so large that it is the bottleneck of the throughput of Internet backbone routers [66]. The routing cache is proposed to be introduced in front of the CAM access, improving the throughput of the routers by bypassing the CAM accesses when it hits. An 1 Tbps line rate throughput for minimum 64 byte packets is achieved [67] with a different cache mechanism.

The work to utilize routing caches on HPC switches is limited [68, 69], and detailed effects on reducing the communication latencies and the performance of HPC applications are not known.

2.4 Job Size of HPC Systems

HPC systems with batch job queuing systems rarely executes parallel applications on all the computation nodes. Instead, they are used with batch job queuing systems, enabling multiple application programs of multiple users to run with a maximum throughput.

Many parallel application jobs are allocated to sub-groups of computation nodes and are executed independently. The maximum computation node size is usually limited to a few thousands, enabling multiple parallel application jobs to be executed in parallel to maximize the job throughput.

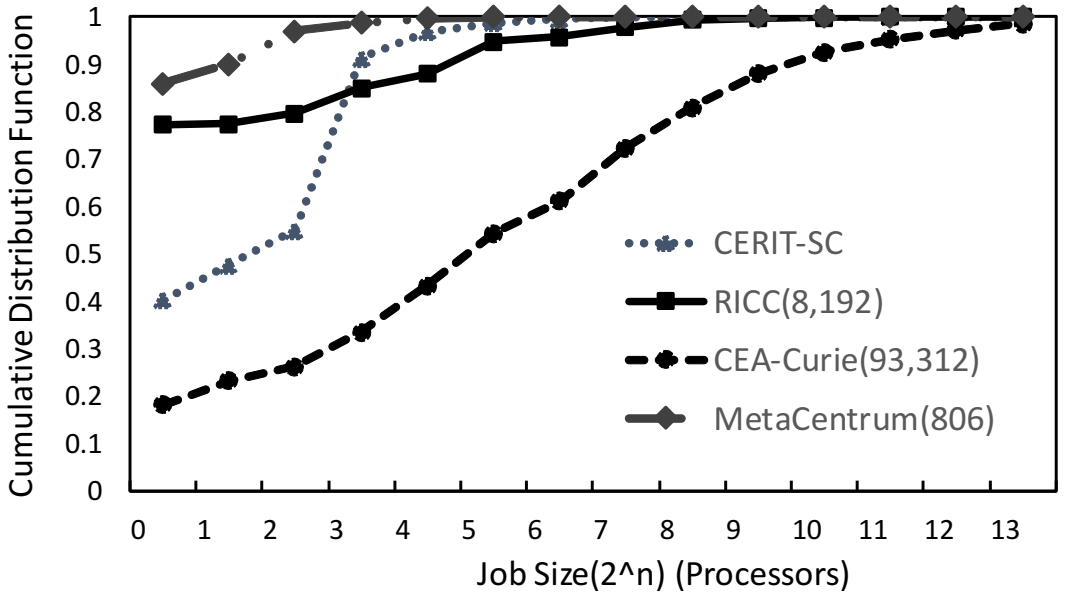


FIGURE 2.13: The Cumulative Distribution of Parallel-Job Sizes.

Figure 2.13 shows the accumulating size of user jobs in CERIT-SC [70], RICC [71], CEA-Curie [72], and MetaCEntrum [73] supercomputers. They are analyzed with the traces of Parallel Workloads Archive (PWA) [74] and [75].

In Figure 2.13, the overall processor numbers are shown in the next of the supercomputer name. Notice that the x-axis shows the processors number, instead of computation node number. All supercomputers have multiple processors in each node, e.g. RICC has 8 processors in a computation node, and CEA-Curie has both CPUs and GPUs in a computation node. Consequently, the computation node number used for each user jobs is far less than the processors number shown in Figure 2.13.

According to Figure 2.13, many user jobs only use a small number of computation nodes in real supercomputers, e.g. 98% of user jobs only use less than 128 processors (32 computation nodes) in RICC supercomputer.

2.5 MPI Traffic Patterns

On a network cache, the access locality is a crucial factor to lead to a high cache hit rate. Parallel computers usually generate unique traffic patterns. Traffic patterns are reviewed in the three layers; point to point communications, collective communications, and traffic pattern of application benchmarks.

MPI programs are dominant for parallel computers program environment. More sophisticated program environment, e.g., PGAS [76] and SHMEM [77], can be employed, and they utilize MPI environment as their runtime.

Figure 2.14 shows a runtime architecture of MPI programs. The figure shows that one or more MPI programs, an MPI library, and an operating system are running in a host compute node. Each user program is running on the MPI library, which is running on the operating system. They have their memory buffers for their own computations dependent on what MPI programs they execute.

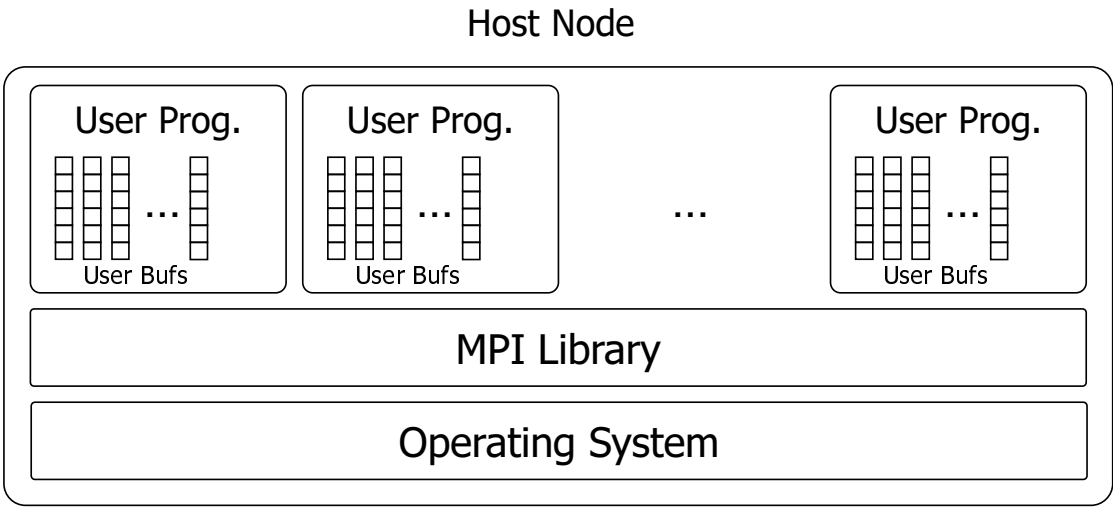


FIGURE 2.14: MPI Programs, an MPI Library, and an Operating System, in a Host Compute Node.

2.5.1 Point to Point Communication

A point to point communication is a message communication between a compute node with another compute node. Each compute node can be mapped logically with one or more communication ranks. Ranks are specified by programmers to MPI applications. Consequently, the traffic patterns of point to point communications are fully dependent on the MPI user programs.

2.5.1.1 Send Communication

A send communication is a message communication which is initiated by a network node. The send communication packet is delivered to another network node, which is called target node. As other network nodes do not receive the communication packet, the send communication is a point to point communication.

Figure 2.15 shows an example of send communication. Six network nodes, rank1, 2, . . . , 6, exist in the figure. Each node has a memory buffer. Assume that the memory content of rank2 is sent to that of rank3. After the content of the buffer of the rank3 has became the same content with that of the rank2, which is described in Section 2.5.1.2, the send communication terminates.

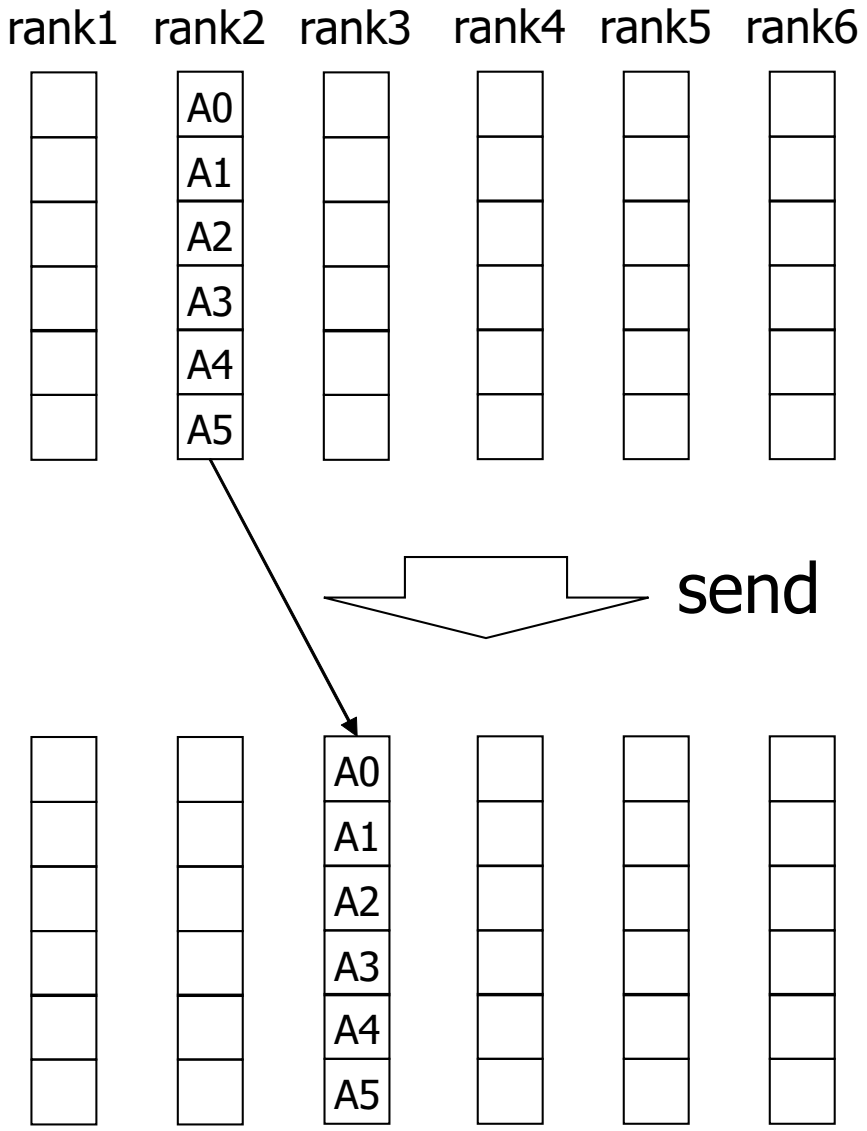


FIGURE 2.15: Send Communication from the Rank2 to the Rank3.

2.5.1.2 Receive Communication

A receive communication is a message communication which is prepared by a compute node. The communication waits until a message from a sender compute node arrives to the compute node.

Figure 2.16 shows an example of receive communication. The rank3 initiate a send communication while the rank1 initiate a receive communication. The actual transmission starts only after both ranks, the rank3 and the rank1 in this example, initiate and be waiting the communication terminates. After the receive communication terminates, the buffer of the rank1 has became that of the rank3.

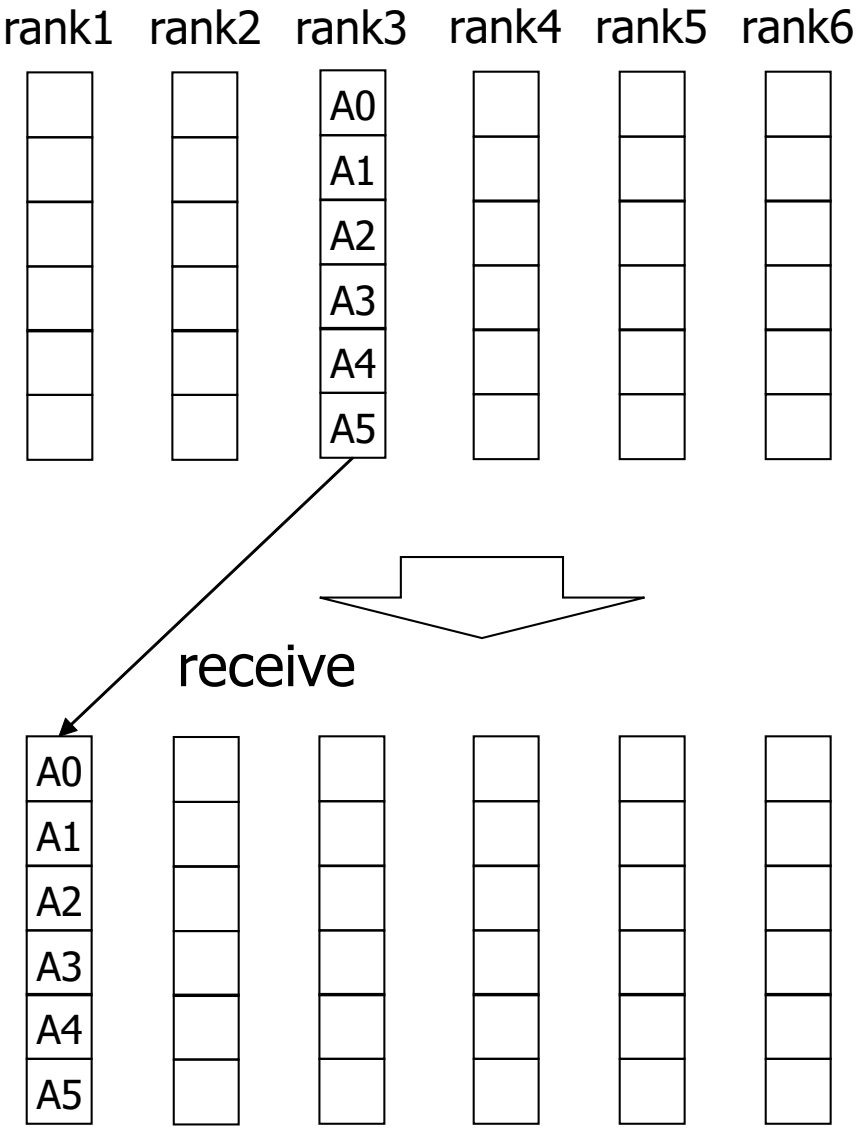


FIGURE 2.16: Receive Communication to the Rank1, from the Rank3.

2.5.1.3 Access Pattern of Point to Point Communication

As the target rank of send communication is specified in each user program, the traffic pattern is fully dependent on the user program.

Same as the send communication, receive communication needs to specify the target rank they receive the communication, the traffic pattern is dependent on the user program.

Thus, the traffic patterns of point to point communication are fully dependent on the user programs As a result, the packet forwarding algorithm of a commodity switch is not able to be optimized to a specific user program.

2.5.2 Collective Communication

Parallel programs usually perform some collective communications. Collective communications involve all compute nodes, different from point-to-point communications. Typical collective communications are explained in this Section.

Since collective communications involve multiple communication ranks, the communication patterns are dependent on the library, instead of user programs themselves. Optimizations of collective communications [78] may be adopted to those libraries.

2.5.2.1 Barrier and Broadcast

Broadcast communication is a communication from a sender node to the rest of all compute nodes. The contents of the buffers in all compute nodes involving the broadcast communication became the same content of that of the buffer in the sender node. As multiple compute nodes involve in the broadcast communication, multiple target node addresses are used. Binomial Tree Algorithm [79] is widely used as an optimized communication algorithm. An optimized scatter-ring-allgather [80] algorithm are used in MPICH communication library [81]. The different algorithms directly affect the traffic access patterns in interconnection networks. Since commodity interconnection networks may take different algorithms on a parallel computer, various traffic patterns may be generated. Thus, we target the worse case, random traffic pattern, in this dissertation.

2.5.2.2 All-to-All Communication

Complete exchange communication is one type of an all-to-all communication. After a complete exchange communication, each rank has a buffer with the content that is a collection of an entry in the same position of all buffers of all network nodes.

Figure 2.18 shows an example of a complete exchange communication. After the complete exchange communication terminates, each rank has all buffer entry corresponding to the number of the rank in its buffer. Traditionally, non-power-of-two process counts

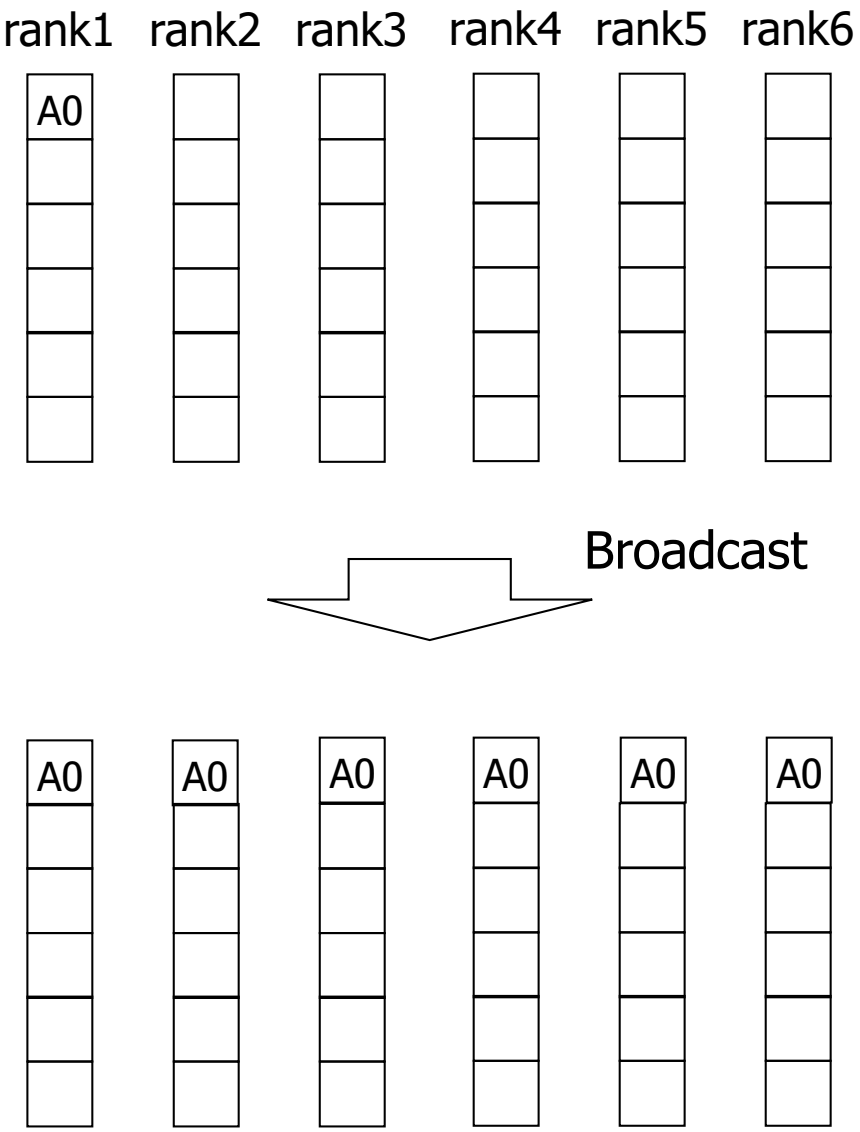


FIGURE 2.17: Broadcast Communication from the Rank1.

(mmsg-npof2) algorithm is used. Similar to the barrier and broadcast, multiple algorithms, which generate different traffic patterns, can be implemented to alltoall.

2.5.2.3 Allgather Communication

Gather communication is an all-to-one communcation, which collects a buffer entry of all ranks to a rank. Allgather communication collects the same data with gather communcation, while all ranks receive the data instead of just one rank as gather communcation.

Figure 2.19 shows an example of an allgather communication. After the allgather communication terminates, all ranks have the same buffer data which is the collection of all ranks.

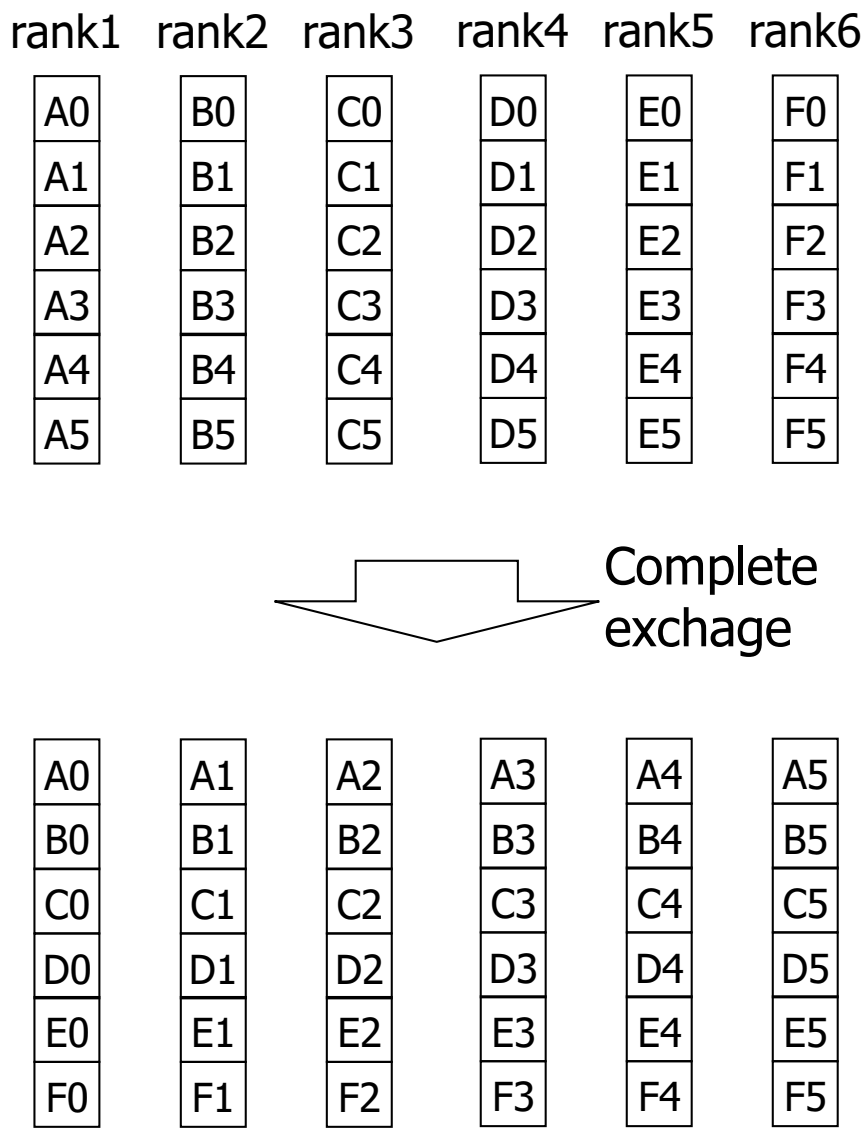


FIGURE 2.18: All-to-All Communication.

2.5.2.4 Allreduce Communication

Reduce communication is an all-to-one communication, which calculates the collection of a buffer entry of all ranks and stores the result to a rank. Allreduce communication calculates and stores the same data with reduce communication to all ranks.

Figure 2.20 shows an example of an allreduce communication. After the allreduce communication terminates, all ranks have the same buffer data which is the results of all calculations of all ranks.

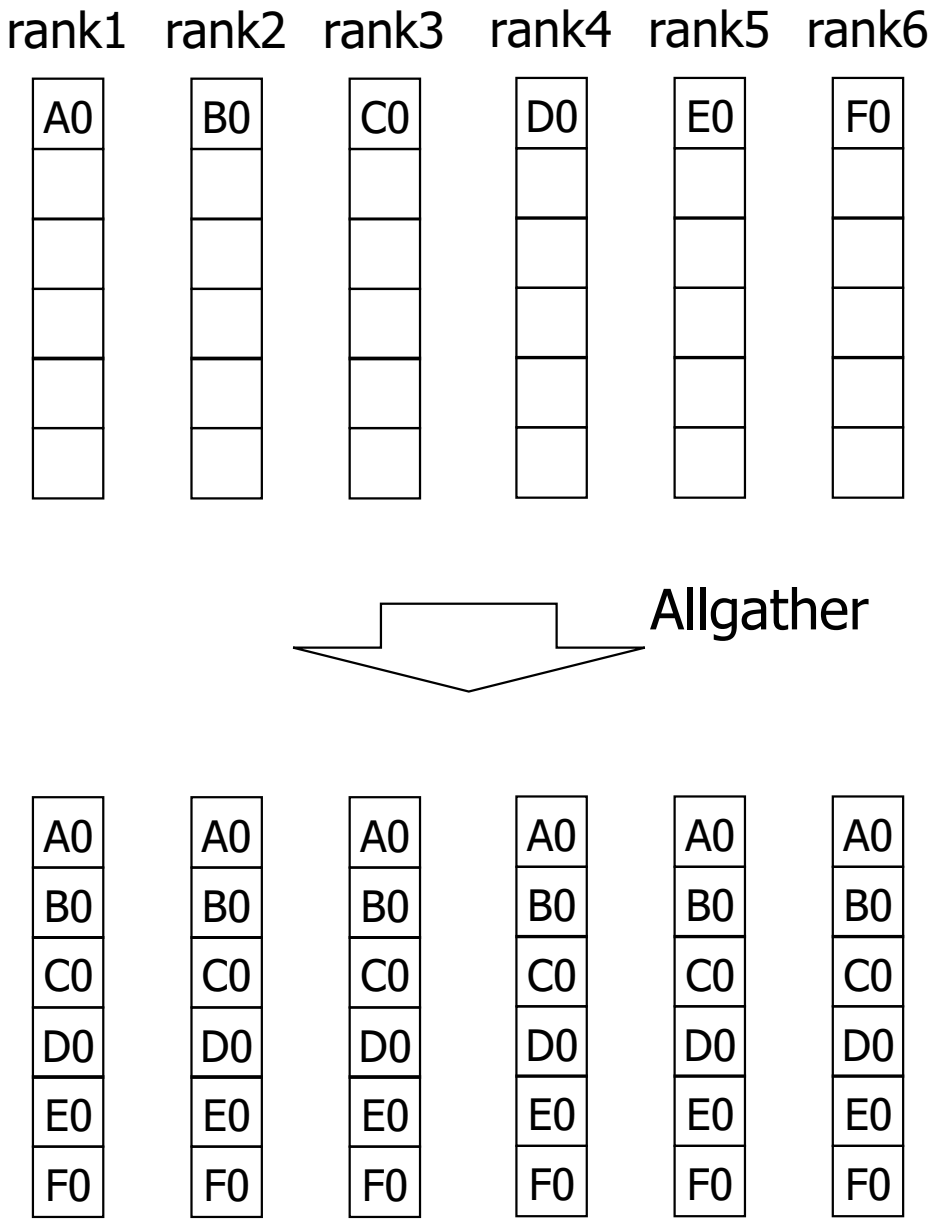


FIGURE 2.19: Allgather Communication.

2.5.2.5 Access Pattern of Collective Communication

Collective communications are initiated, transferred, and terminates in MPI libraries, instead of user programs. Each MPI collective communication library calls are implemented with one or more collective communication algorithms to accomplish the specified collective communication. The communication algorithms of collective communications are dependent on the implementations of MPI libraries.

Communication algorithms is able to be selected statically by the user, by command-line

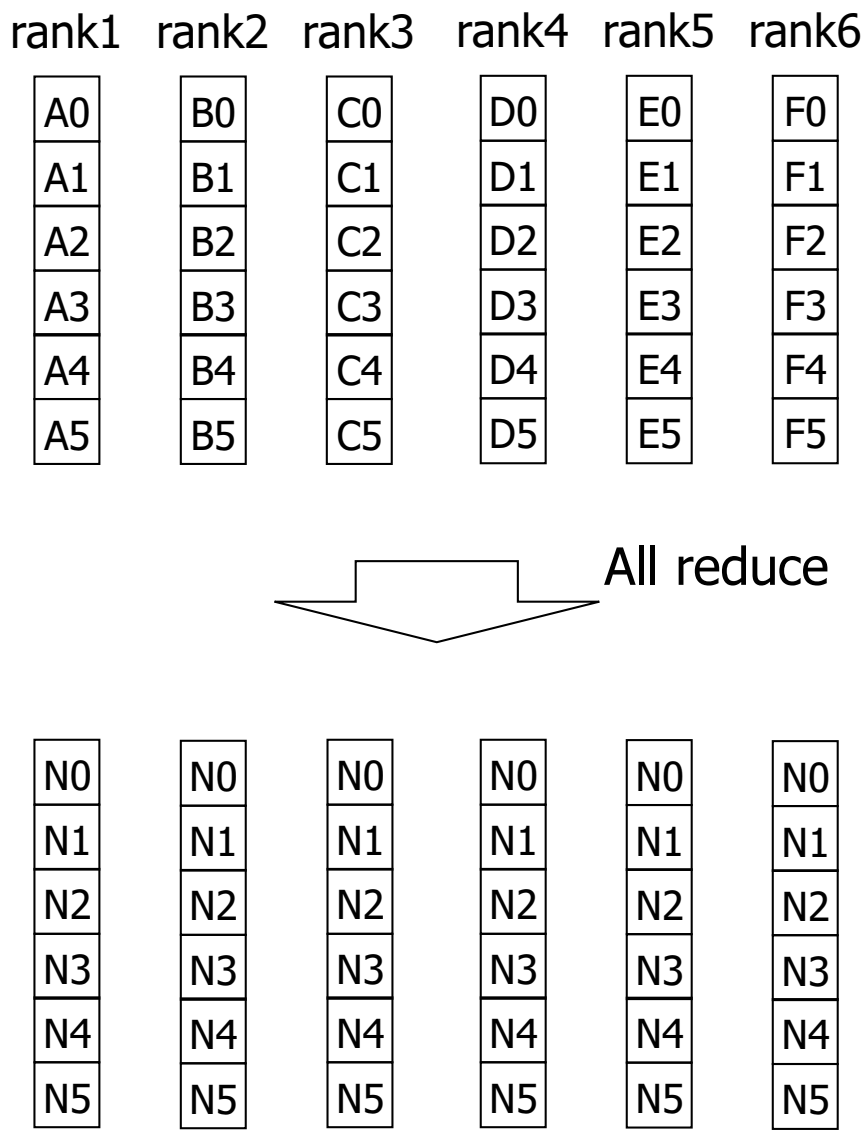


FIGURE 2.20: Allreduce Communication.

options or configuration files of the MPI library. The working algorithm is even automatically selectable dependent on the network sizes and the communication sizes [82]. Thus, the traffic patterns of collective communication are independent on the user programs and the network topologies. As a result, to optimize the packet forwarding algorithm of a commodity switch can suffer from low communication performance because of the multiple possible collective communication algorithms.

2.5.3 Access Patterns of Parallel Applications

Cache works well when there exists a strong access locality of the packets. Unfortunately, there are a large number of collective communications on typical parallel applications,

that generate a global traffic access. For example, it was reported that Mira supercomputer executed many jobs in which significant part of the execution time is collective communication [83]. The most frequently used collective communications are allreduce, bcast, alltoall, and barrier, which are explained in the previous subsection. They fully utilize all compute nodes, or generate a global traffic access to compute nodes within a job.

Fabien and Koibuchi analyzed the traffic patterns of NAS Parallel Benchmarks [84] by the network simulation [85]. In all simulations, they assumed that the interconnection network consists of 64 switches with 64 processes. The considered applications are the Fourier Transform Class A (FT), the Integer Sort Class A (IS), the Block Tri-diagonal solver Class A (BT), the Scalar Penta-diagonal solver Class A (SP), the Lower-Upper Gauss Class B —due to benchmark restrictions— (LU), the Conjugate Gradient Class A (CG), the Data Traffic Class A (DT), the Embarrassingly Parallel Class A Class A (EP), and the Multi-Grid Class A (MG) from the NAS benchmarks. They also ran the Memory Multiplication (MM) provided by SimGrid, the Himeno benchmark Class S and the replicated version of the Graph500 MPI program.

Results in [85] showed the average traffic for all benchmarks, between a source-and-destination pair. We found that there are a large number of local traffic in four benchmarks (BT, SP, LU and HIMENO). By contrast, uniform traffic is found in FT, IS, MM and Graph500.

Through the facts of the prior works [83, 85, 86], our packet forwarding cache design assumes that the worst case in which the communication pattern is less locality (uniform).

Chapter 3

Problem Statement

Interconnection network architecture for both supercomputers and data centers are converging, due to the heavy data centric workloads. While high-bandwidth and low-latency network performance are critical for supercomputers, interoperability with commodity standards is inevitable for data centers. Cray Slingshot interconnection network [87] provides key features for both supercomputers and data centers. It allows the interconnection network to be interoperable with standard Ethernet switches, while providing the high-bandwidth and low-latency network performance for supercomputers.

Network switch is a heart of an interconnection network, and its aggregate network throughput increases year by year. Switch delay dominates communication latencies in interconnection networks. At a conventional switch, routing decision imposes a significant delay. Reducing the access latency is crucial for the upcoming low-delay switch in parallel computers. Regarding switch ASICs, Broadcom releases the design of Tomahawk 3 Ethernet switch 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 switch ASIC to the HPC domain is a Rosetta switch [70] in Cray Slingshot interconnection network which uses Dragonfly network topology. Rockley Photonics also illustrates a hyperscale switch. RANOVUS Odin plans to utilize a siliconphotonics engine in a 51.2 Tbps switch. Continuing to increase the switching capacity of a switch ASIC relies on optical integration.

Intel and Barefoot provide a 12.9 Tbps P4-programmable Tofino2 switch ASIC, which is one of the highest packet throughput switch ASICs, whose block diagram is shown in

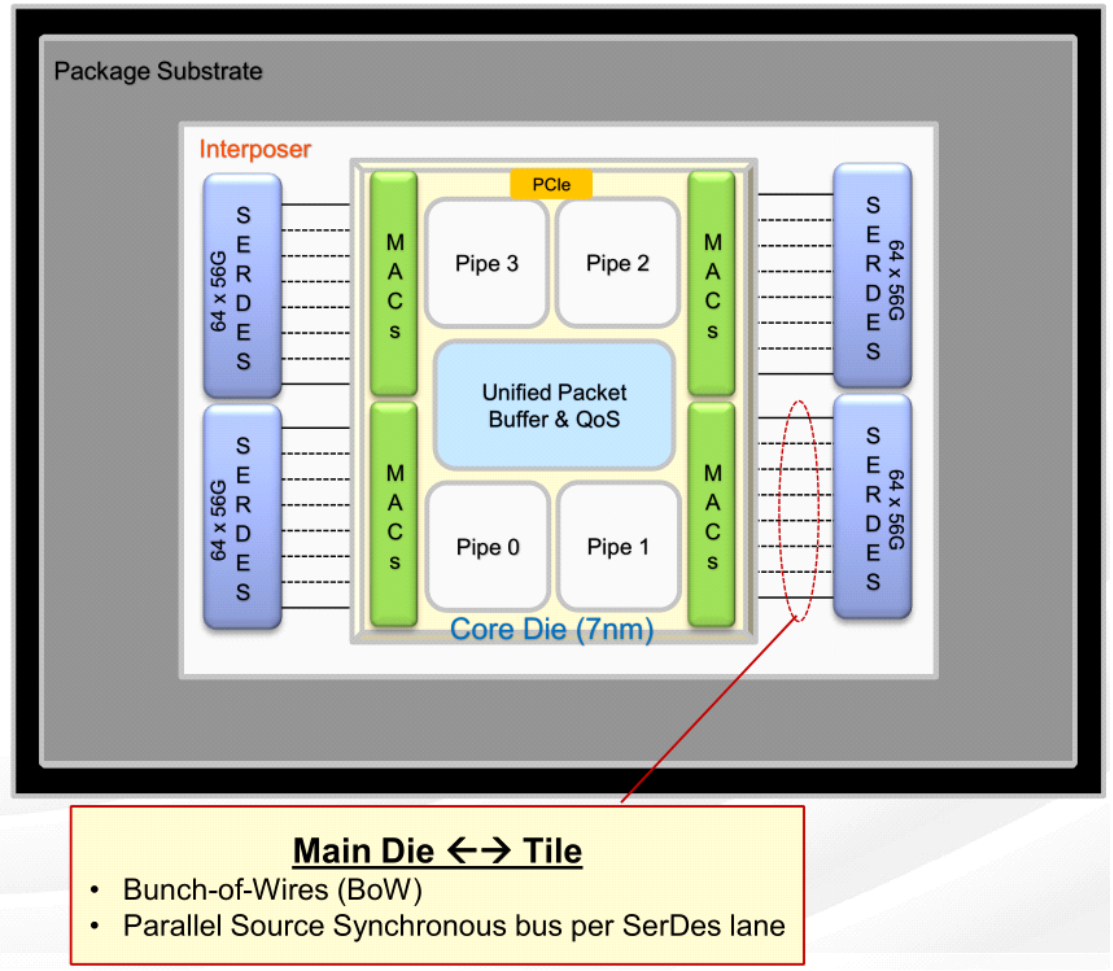


FIGURE 3.1: An Example of a Switching ASIC; Intel Tofino2 [4].

Figure 3.1. In the figure, four packet forwarding pipelines "pipe" are implemented in the 7nm switch die. 64 MB Unified Packet Buffer is used to store the forwarding packets. Four 400G Media Access Controller(MAC)s are connected with 260 lanes of 56G-PAM4 SerDes Tiles. 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. The 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 switching capacity with 3.3 Bpps to 9.52 Bpps packet forwarding rate, resulting in only 2.1 Tbps to 6.2 Tbps throughput for a minimum packet length. 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 2.3.2. The gap between switching capacity and

forwarding packet rate will increase even when large TCAM will be integrated into 2.5D chip implementation.

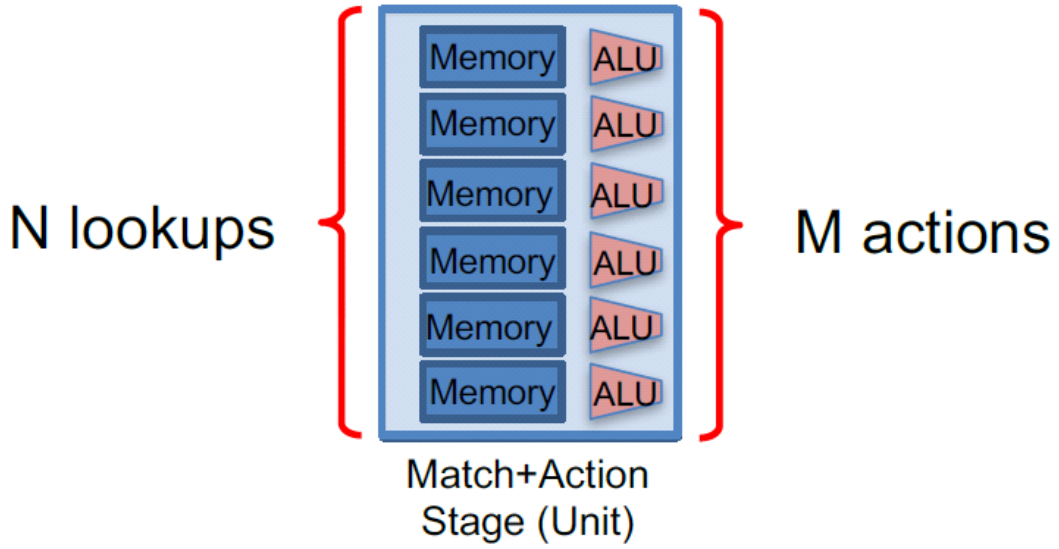


FIGURE 3.2: Simultaneous Parallel Lookups and Actions [5].

There are switch ASICs that implement simultaneous parallel assessing to the TCAM. Figure 3.2 shows an example of such a switch [5] that supports multiple simultaneous lookups and actions. When N lookups come to the ASIC, the six TCAM memory tables handle them, as shown in Figure 3.3. Then, multiple results are returned to the switch ASIC, resulting in multiple times more results than a single TCAM chip offers. However, the huge total energy consumption of TCAM chips limits the number of the parallelism, as a large part of forwarding logic is made up with TCAMs [4]. Consequently, the overall packet forwarding rate with multiple TCAM accessing is limited.

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 Chapter 1. The key design to resolve the problem of the packet forwarding performance is a packet forwarding cache architecture explored in this dissertation. More precisely, "address patterns" of interconnection networks should be found, and the packet forwarding cache architecture should be optimized for enjoying the address pattern.

One might think that a function to route packets should be simplified by dropping routing tables from a switch. That is, source-routing implementation should be employed instead of distributed routing, to increase the packet forwarding throughput. This is technically true, and historically Myrinet2000 supports source routing. This concerns the network design philosophy; intelligent switch with simple network interface or simple switch with intelligent network interface. As soon as the network designer selects the former, we cannot be free from the distributed routing, and the routing cache is currently

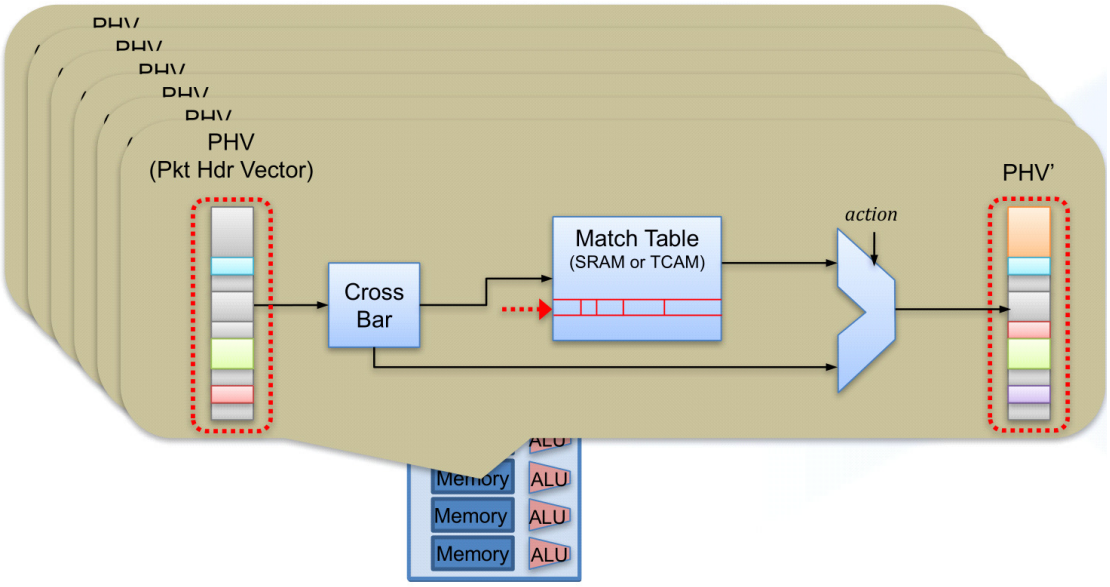


FIGURE 3.3: Multiple TCAMs to Support Simultaneous Parallel Accesses [5].

only one solution for high-bandwidth low-latency switches. Since current commodity interconnection technology continues to take the former, the routing cache will be a valuable.

Chapter 4

Packet Forwarding Cache

A switch ASIC should be widely used in parallel computers and high-end datacenters. To improve the forwarding capacity and throughput of a network switch, a packet forwarding cache design for arbitrary network topologies is presented in this chapter.

4.1 Scope and Assumption on Job Size

In this chapter, jobs of up to 2K computation nodes are considered. This assumption is reasonable, and its reason is explained in Section 2.4.

The total number of the target addresses of all packets in a user job is less than or equal to the number of computation nodes within the job. Most jobs are small on average, and $2K$ table entries are enough for them. It is reasonable to support such a job size at the packet forwarding cache. In this chapter, the case for the worst-case design for up to 2K-computation node jobs is considered.

4.2 Packet Forwarding Cache Switch Architecture

4.2.1 Outline

Figure 4.1 illustrates our target switch with the packet forwarding cache. Later, we call it the cache switch. It is the same as the baseline switch in Figure 2.6 except for the

routing-computation unit.

After an incoming packet is stored at an input port, its output port information is obtained via a routing-table lookup in the traditional switch. Routing tables are Forwarding Table in InfiniBand and MAC address table in Ethernet. The entry size is significant because it needs to input packet address information and output the port information where the packet need to transmit. For example, Mellanox MTS3600 (36 ports, 40Gbps InfiniBand Switch) has 48K entries, and Mellanox SN3000 Series datacenter switches (32 to 48 ports, 400Gigabit Ethernet(GbE)) have 512K entries. As a result, routing tables are implemented using CAM(Content Address Memory) instead of SRAM.

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 destination of the packet, 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.

When the cache hits, the switching latency can be reduced because no CAM access occurs for the routing. On the other hand, the switching latency increases when the cache misses because the CAM access is needed in addition to the latency to accessing the cache.

Current 400Gbps Ethernet Standards are achieved with multiple lanes, due to the limitation of single SerDes lane speed. E.g. 400GBASE-FR8 is a combination of eight 50Gbps lanes with wavelength-division multiplexing (WDM).

When WDM or DWDM(Dense Wavelength Division Multiplexing), which is used with 400G-ZR, is used for the link, the cache can either be installed to each link or each wavelength, depending on the packet throughputs they provides. E.g. if the cache switch architecture is fast enough to handle the whole packets for the link, a switch link architecture with only one cache can be selected for smaller chip area overhead, whereas when the link throughput is higher than the maximum throughput of the cache switch architecture, the cache can be implemented for each wavelength for higher overall throughput.

4.2.2 Function of the Packet Forwarding Cache

The function of switches with the packet forwarding cache is the same, except the routing computation stage, as the baseline switch microarchitecture shown in Figure 2.6. The

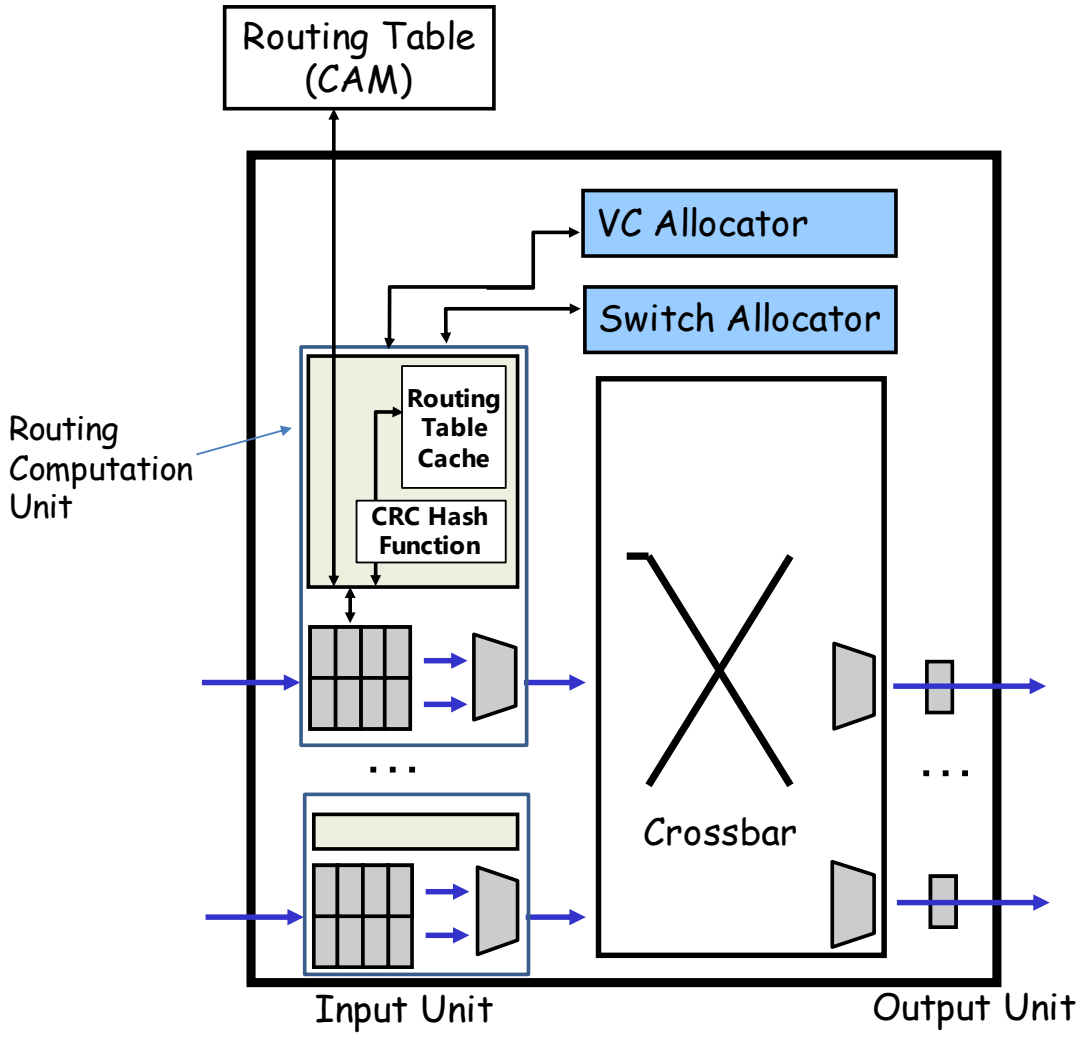


FIGURE 4.1: Block Diagram of Switch with the Packet Forwarding Cache.

routing computation stage of a switch with the packet forwarding cache is described in this section.

Figure 4.2 shows the microarchitecture of a switch with the packet forwarding cache, which is proposed in this Chapter. The switch has an additional packet forwarding cache mechanism in addition to the baseline switches. All input ports of the switch has a packet forwarding cache. This is because, in general, packets to a same target address may be routed to different output ports.

Packets arrived to the input ports of the baseline switch are stored in the channel buffer. Then, the target output port information are derived by accessing the off-chip CAM. In the case of the switches with the packet forwarding cache, the packet forwarding cache of the input port is accessed after a packet is arrived and stored to the channel buffer. A cache line is selected by a hash number calculated with the target address node information of the input packet header. If the tag of the selected cache line is the

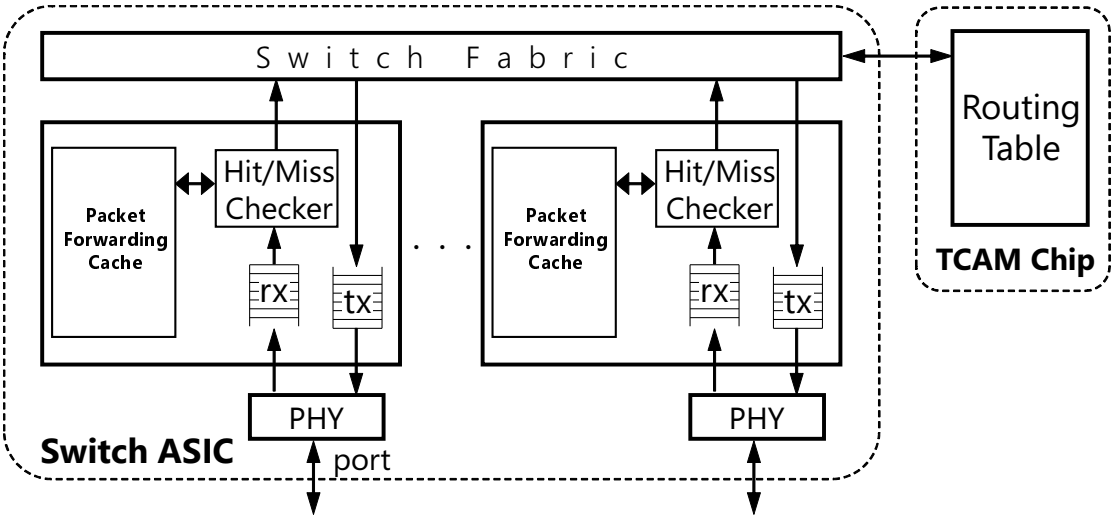


FIGURE 4.2: Relationship between the Packet Forwarding Cache and the TCAM.

same to the target node address of the input packet, the access to the packet forwarding cache is a hit, and the output port can be derived from the cache. On the other hand. if the access to the packet forwarding cache is a miss, the output port information is derived by accessing the off chip CAM. Therefore, the access to the off chip CAM is skipped if the cache hits. Note that the CAM is assumed to be large enough to store entire target address space of the network.

The routing computation stage of the switch with packet forwarding cache is further divided into three pipeline stages as follows.

1. Hash value calculation
2. Accessing the cache, and
3. Accessing the CAM.

Each stage takes a few cycles. The stage of the hash value calculation can be finished in one cycle by CRC(Cyclic Redundancy Check). The stage of accessing the on-chip cache takes one to a few cycles, which is dependent on the cache capacity. In this chapter, a small size on-chip cache, e.g. 24KiB, is assumed, and the access latency is 0.41 nano seconds, which is simulated result with CACTI6.5 [88] using 32 nm design process. This access latency is small enough to fit in one cycle of a switch ASIC operated in less than 1GHz, even the latency to determine whether the access is a hit or a miss is included. The CAM is accessed only when the cache access is a miss, and an additional 25 nano seconds latency, which is 25 cycles in a switch operated in 1GHz, is needed.

4.2.3 Packet Forwarding Cache on Topology Changes

A switch with routing tables can be used in different topologies by only reprogramming the table content [24]. Although the changes of network topology for HPC switches rarely occur, they are certainly allowed in the commodity network standards. To strictly following the network standards in the switch with packet forwarding cache, the cache content is fully flushed on network topology changes.

The functions of the switch with the packet forwarding cache on topology changes are described as follows.

1. Detect the change of the network topology.
2. Flush and temporally disable the packet forwarding cache.
3. Reprogram the routing table (CAM) to support the new network topology.
4. Enable the packet forwarding cache.

As a result, consistent operations for data hazard, e.g., WAW (Write After Write), RAW, and WAR, are not necessary for the packet forwarding cache. The restart of the packet forwrding cache sould suffer from initial cache misses.

4.2.4 Support for Routing Options

Our cache switch can support adaptive routing, which is described in Section 2.2.2. The behavior of a conventional switch for adaptive routing is the same as that for deterministic routing except that the routing computation (RC) returns multiple output candidates, and switch allocation request to reserve an output ports from the output candidates. To support adaptive routing, the size of a cache entry slightly increases to store the information of multiple output ports.

4.2.5 Cache Architecture

4.2.5.1 Data structure

In this dissertation, each entry of the packet forwarding cache is a 12 bytes line, consists of 8 bytes target node address descriptor as a cache tag, 2 bytes output port descriptor as a cache data, and 2 bytes reserved data which can be used by QoS control.

4.2.5.2 Cache hierarchy

The size of the packet forwarding cache is large enough to store all the computation node addresses used in most jobs. Therefore, capacity misses of the packet forwarding cache will not occur for communications in such a job. Hence, in this Chapter, a large second level shared cache which is able to be used among all input ports of switches is not adopted. Each input port has its own packet forwarding cache and can access it exclusively.

4.2.6 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.

The legend represents the number of packet destinations. 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.

4.3 Evaluation

4.3.1 Cache Size and Latency

The cache area and latency is estimated 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. A four-way set-associative is assumed.

Figure 4.3 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.

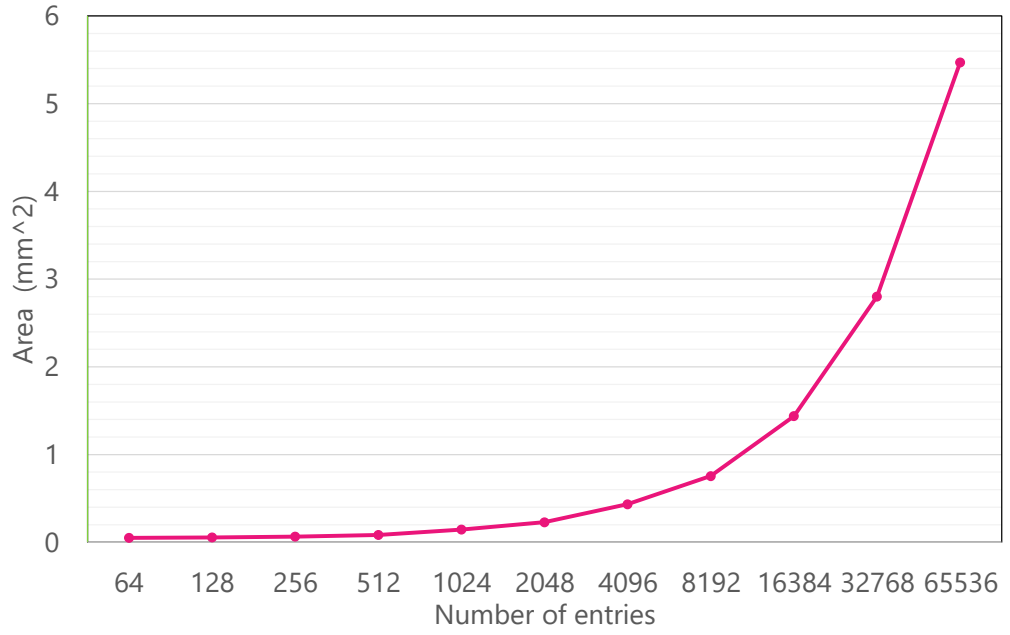


FIGURE 4.3: Cache Area Overhead at an Input Port.

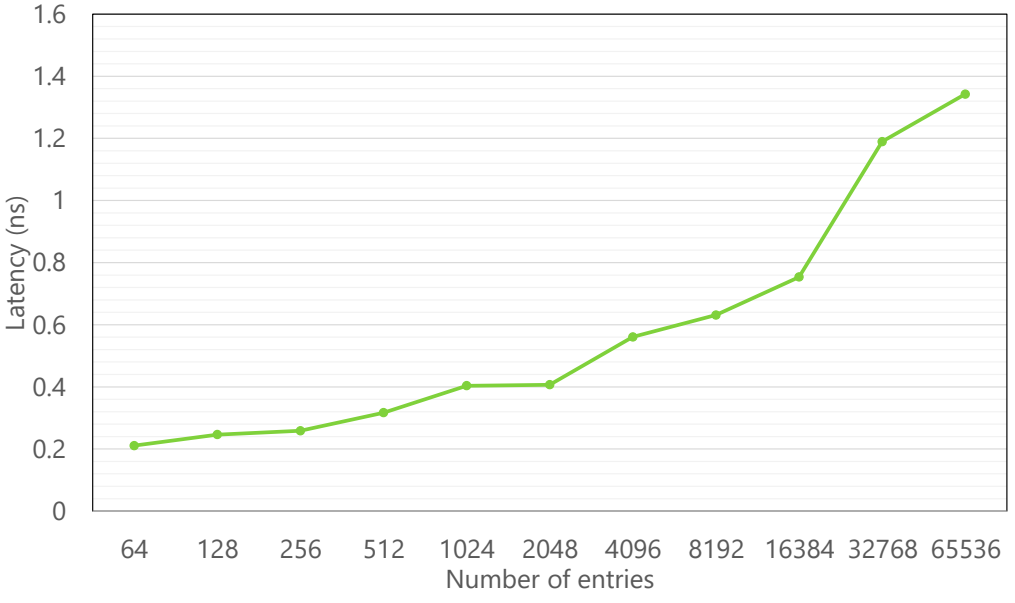


FIGURE 4.4: Cache Latency Overhead at an Input Port.

Figure 4.4 illustrates the latency overhead of the cache. When 2,048 entries are used, the latency is 0.41 nanoseconds. Ideally, the packet forwarding throughput becomes 1.64 Tbps (1 / 0.41 × 84 × 8).

4.3.2 Compulsory Miss Rates of the Packet Forwarding Cache

Cache misses can be categorized as compulsory misses, initial misses, and conflict misses. Capacity misses rarely occur to a packet forwarding cache with enough entries in a small size computation node systems. Compulsory misses are negligible by kicking warm-up communications among all computation nodes on the job starting up. Therefore, conflict misses of packet forwarding caches are evaluated in this section. Since conflict misses are highly dependent on their associativity, the conflict miss rates and associativity of the packet forwarding cache are evaluated.

4.3.2.1 Evaluation Conditions

The capacity of the packet forwarding cache is 24 KiB(2,048 entries), which is about the same as the capacity of general CPU L1 caches, with considering the capacity miss overhead and chip area overhead. With this capacity, no capacity miss occurs in a job less than 2,048 computation nodes. The area overhead of the packet forwarding cache is evaluated with CACTI6.5[88]. Only 0.18 mm^2 and 0.73 mm^2 area overhead for each port is needed for high performance transistor model, 32 nm and 65 nm process technology, respectively. The area overhead is only about 2%, or 7.3 mm^2, for 10 port packet forwarding caches of the ICC chip(10 ports, Fujitsu Semi-Conductor, 65 nm, 329 mm^2 [89]), which is used for the K computer. Besides, the area overhead of CRC hash calculation hardware is negligible, e.g. hundreds μmm^2 [61].

An uniform traffic pattern, which is uniform for the all target address and the calculated hash number, and communication traces of NAS parallel benchmark 3.3.1(MPI) are used. The problem size is Class A, and the rank size is 256, for the NAS parallel benchmarks. The network topology used for the NAS parallel benchmark evaluation is 3D torus(4×4×4), with each computation node connecting to a switch.

4.3.2.2 Uniform Traffic Results

Figure 4.5 shows the software simulation results of conflict miss rates (Y-axis) corresponding to each associativity (X-axis) of the packet forwarding cache.

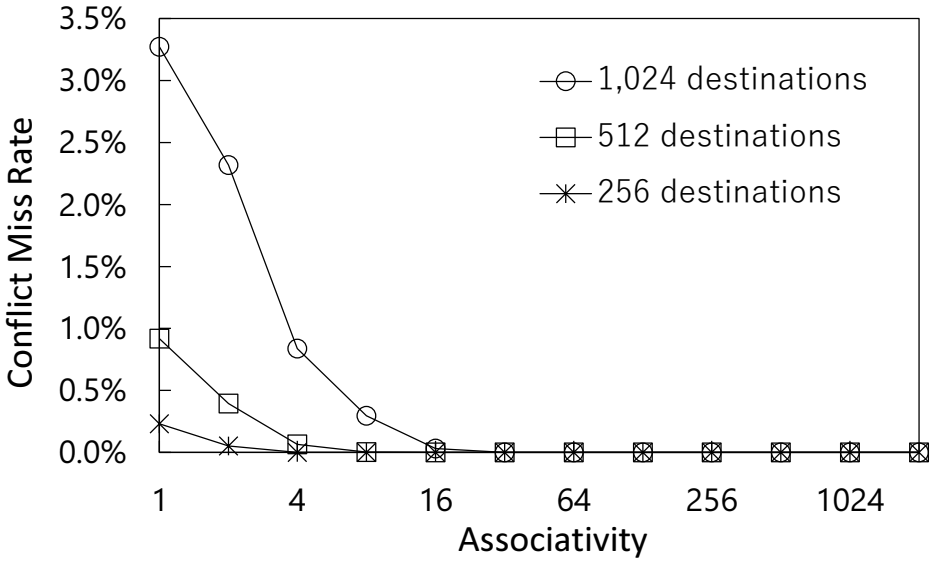


FIGURE 4.5: Associative Number and Ratio of Entries Replaced by Conflicts (Uniform Traffic).

From the simulation results, the conflict miss rates is less than 0.1% for the case of inserting 512 targets to a 4-way set associative cache and the case of inserting 1,024 targets to a 16-way set associative cache. Hence, conflict misses rarely occur for 1,024 computation nodes jobs with 16-way set associative cache. Because the sizes of more than 90% jobs used in many HPC systems are less than 1,024, as shown in Figure 2.13, the timing overhead of conflict misses of the packet forwarding cache is negligible for most jobs.

4.3.2.3 NAS Parallel Benchmarks Results

Figure 4.6 shows the conflict miss rates of each associativity of the packet forwarding cache by executing NAS parallel benchmarks.

No conflict miss occurs on associativity 2 and 4 caches for all applications. Accordingly, the hit rates are 100.0% on the caches with associativity 2 and 4, except the compulsory misses. Besides, all applications except DT show extremely high 99.7% to 100.0% cache hit rates even including compulsory misses on the cache with associativity 4. (The cache hit rate of DT including compulsory misses is 95.6%, because of its small number of communications.) The results illustrate that the four-way set-associative cache is reasonable for almost avoiding the conflict cache miss.

We analyzed 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.

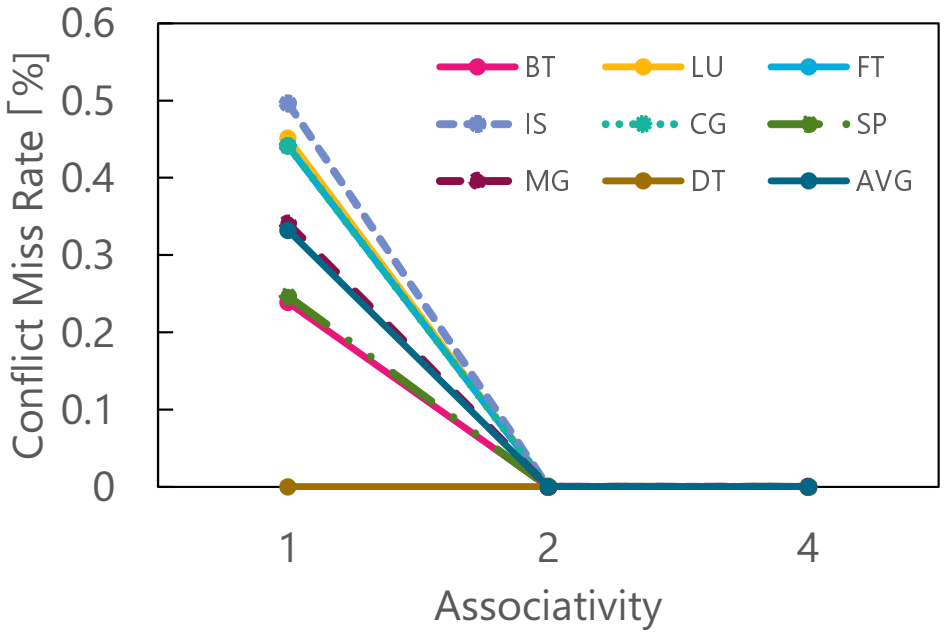


FIGURE 4.6: Associative Number and Ratio of Entries Replaced by Conflicts (NAS Parallel Benchmarks).

4.3.3 Parallel Applications Performance

In this section, the execution performances of HPC parallel applications are evaluated, comparing the performances on a baseline interconnection network without the packet forwarding cache and an interconnection network with the packet forwarding cache.

4.3.3.1 Simulation Conditions

SimGrid v3.25 [90] is used to evaluate the execution performances of applications. Two parallel computation systems are used for the evaluation, and their environmental parameters are shown in Table 5.2.

The routing latency is 10 nano seconds [91] for the baseline switch without the packet forwarding cache. Other switching latency is set to 50 nano seconds. The 50 nano second latency is the sum of the latency of output buffer allocation, the latency of crossbar allocation, and the latency of transmitting the packet from the input port and the output port. Consequently, the minimum latency to forwarding a packet in the switch is 60 nano seconds.

The routing latency of the switch with the packet forwarding cache on a cache hit is 1 nano seconds. This 1 nano seconds latency is a sum of hash value calculation and cache access latency, on an 1 GHz switching chip.

The routing latency on a cache miss suffers from an additional 10 nano seconds latency, which is the same as the access latency to the baseline switch without the routing table cache. Accordingly, the access latency of the switch with the packet forwarding cache is 51 nano seconds on a cache hit, and 61 nano seconds on a cache miss, respectively.

Each computation node has 50 TFlops computation power. The network topology is random [92–94], and its degree is set to eight. The other network topologies are evaluated in Chapter 5.

NAS parallel benchmarks 3.3.1(MPI) are used. The problem size is Class B, except that IS and FT is Class A, because of the simulation time limitation. The capacity miss rate and conflict miss rate on executing the NAS parallel benchmarks are both 0% for 4-way set associative packet forwarding cache, from the previous Section. Compulsory misses can be avoided by all-to-all commutations before starting the job. Therefore, no cache miss is assumed to occur in this evaluation.

4.3.3.2 Evaluation Results

The evaluation results of NAS Parallel Benchmarks are shown in Figures 4.7 and 4.8. Link bandwidth is varied from 100Gbps to 1.6Tbps. The Y-axis shows relative Millions of Operations Per Second(MOPS), which is 1.0 on the conventional switches (Conv. Switch) without the packet forwarding cache. The larger the value of MOPS is better. From the evaluation results, the MOPS performances of NAS Parallel Benchmarks are improved by the switch with the packet forwarding cache(Switch w/ Routing Cache). With NAS Parallel Benchmarks, the amount of computation decrease for each computation node as the process number increases. Thus, the communication latency affects more on the execution performance for a system with more processes and larger network size.

NAS Parallel Benchmarks have different traffic patterns [85]. For example, FT, IS, and MM conduct many all-to-all communications and traffic among all network nodes, while BT, LU, and SP conduct many neighboring communications. As a result, the former group benchmarks represents higher performance improvement, more than 4.0x at least, than the latter group benchmarks, less than 3.0x at most.

Note that the evaluation results show that the average performances of all parallel benchmark applications have been improved with the switches with the packet forwarding cache. Consequently, the effect of the packet forwarding cache to improve network throughput and line rate are significant. More exactly, it achieved 4.39x maximum performance improvement on 256 switches and 2.76x on 32 switches on average.

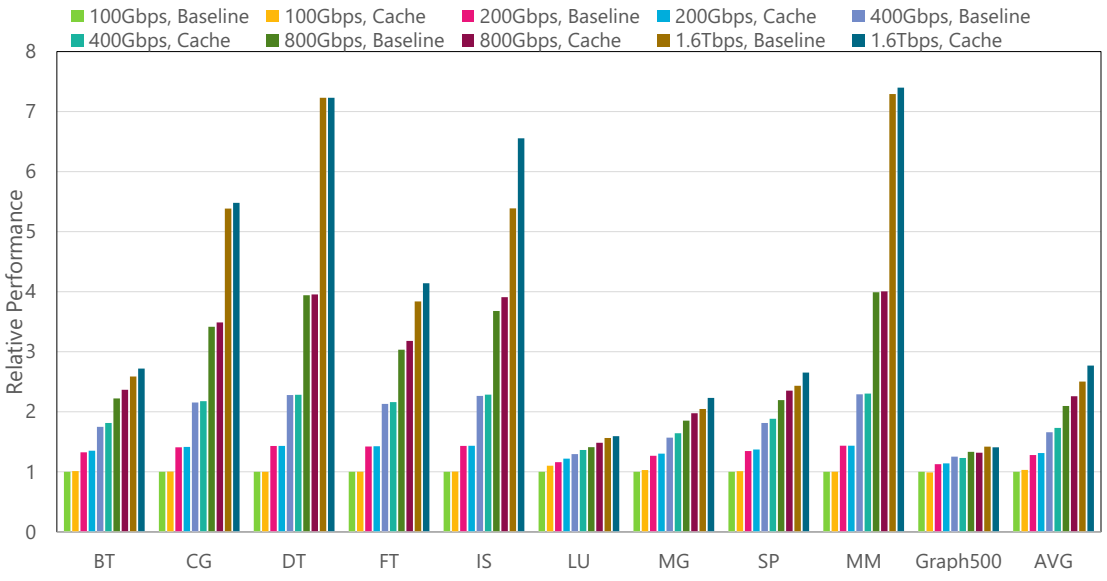


FIGURE 4.7: Relative Application Performance (32 Switches, Random).

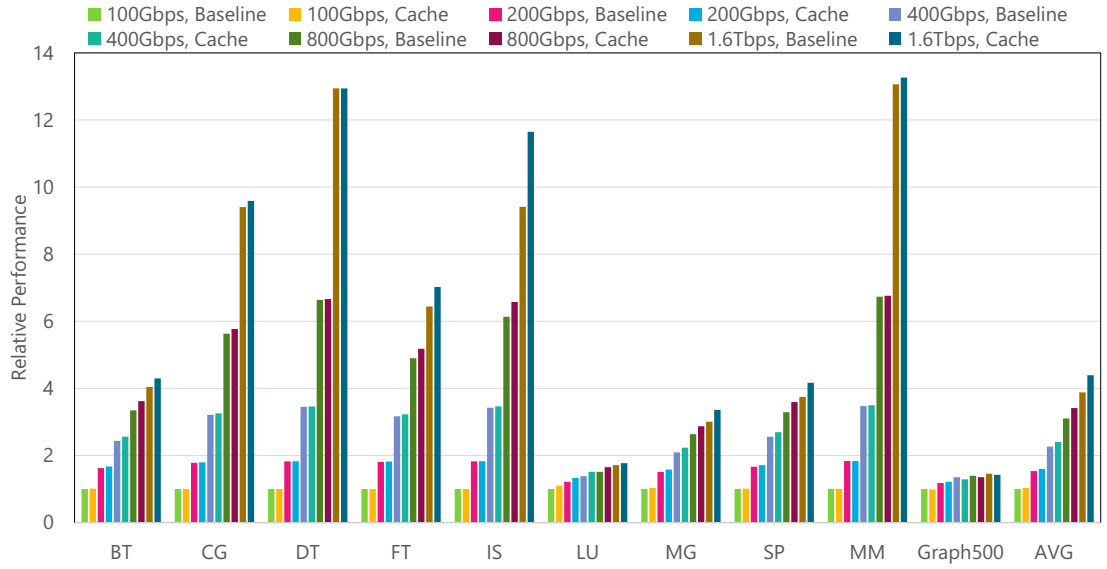


FIGURE 4.8: Relative Application Performance (256 Switches, Random).

4.3.4 Zero-Load Communication Latency Analysis

In this section, performances of large-scale networks is analyzed. SimGrid, which is used for the simulations in the previous Section, is not feasible to simulate networks with more than 256 computation nodes. Therefore, the effect of the packet forwarding cache on a large-scale network, from a few thousands nodes to 30 thousands nodes, is analyzed. Analysis results [95] on interconnection network is reported to be consistent on the simulation results of interconnection network simulators. The performance analysis results are able to be used to determine system parameters of large-scale systems at the initial stage.

4.3.4.1 Analysis Setup

In this analysis, jobs are allocated to a rectangular area of computation nodes of K-ary N-cube torus which consists of K^N computation nodes, assuming K to be an odd number. Each computation node is connected to a switch. Dimensional routing is used. The cases where multiple computation nodes are connected to a switch, or K is an even number, are also able to be analyzed with the same method.

Every input port has a packet forwarding cache. The size of each packet forwarding cache is M entries.

Every computation node produces communication packets with the intervals following the Poisson distribution. Every communication packet is transmitted to the network in the order they are produced. The target address of the communication packet is determined randomly from all computation nodes when it is produced.

In this analysis, different from the evaluation environment used in Section 4.3.2 and Section 4.3.3, capacity misses occur because of the large scale network and affect the interconnection network performance. On the other hand, no conflict miss occurs because random traffic and enough associativity is assumed. In addition, no compulsory miss occur because a warmed up network is assumed.

4.3.4.2 Cache Hit Ratio

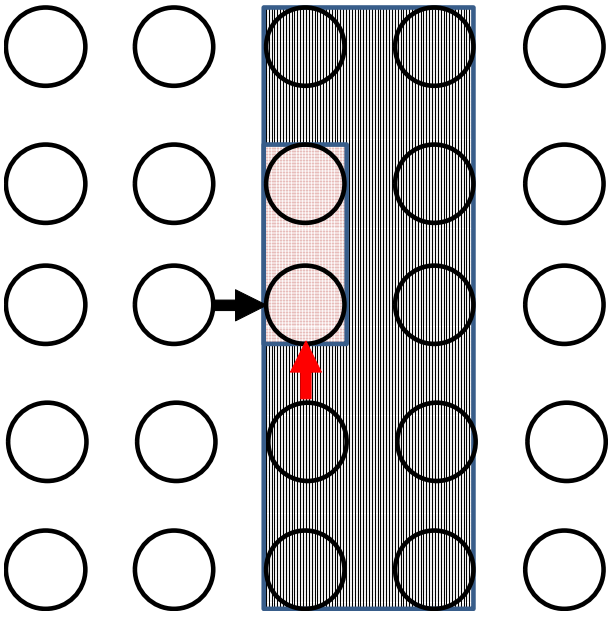


FIGURE 4.9: Example of 5-Ary 2-Cube Torus.

Figure 4.9 marked possible destination nodes of a packet from x dimension port and y dimension port in a 5-ary 2-cube torus network. A packet comes from the x dimension input port, which is described with a black colored arrow in the Figure, is transmitted to one of $K\lfloor K/2 \rfloor$ target nodes group, which is shown in the black rectangular. A packet comes from the y dimension input port, which is described with a red colored arrow in the Figure, may be transmitted to the $\lfloor K/2 \rfloor$ target nodes group, which is shown in the red rectangular. A packet comes from the local computation node can be transmitted to $K^N - 1$ target nodes, which is all nodes except the node produces it.

Assuming a random traffic pattern, the hit ratio of the i dimension input port $P_i (0 \leq P_i \leq 1)$ in a K-ary N-cube torus network can be calculated from the possible number of target node of the input port with Formula 4.1. The dimension of the input port from the local computation node is defined to be 0.

$$P_i = \begin{cases} t_i & (t_i < 1) \\ 1 & (otherwise) \end{cases} \tag{4.1}$$

$$t_i = \begin{cases} \frac{M}{K^N - 1} & (i == 0) \\ \frac{M}{K^{N-i}\lfloor K/2 \rfloor} & (0 < i) \end{cases} \tag{4.2}$$

4.3.4.3 Zero-Load Communication Latency

Let C be both link latencies from a computation node to a switch and between two switches, S_{base} be a switching latency on a hit to the packet forwarding cache, and S_{pnl} be a penalty on a miss to the cache.

A zero-load communication latency $L_{i,j}$, which is the time of a packet which comes from i dimension input port of a switch to arrive to the input channel of a switch in j hops away, can be described below.

$$L_{i,j} = j(S_{base} + S_{pnl}(1 - P_i) + C) \tag{4.3}$$

The maximum zero-load communication latency L_{max} is equal to the latency to a node in $\lfloor K/2 \rfloor$ hops away along the dimension of K-ary N-cube torus. As the cache hit ration is dependent on the dimension of input ports of a switch, L_{max} can be described below.

$$L_{max} = L_{0,1} + \sum_{i=1}^{N} L_{i,\lfloor K/2 \rfloor} + C \tag{4.4}$$

Note that, in Formula 4.4, the hop number of a communication packet comes from the local computation node, dimension 0, is 1. As a result, the hop number of a packet from a source node to a destination node is 1 more than the number of switches along the communication route.

4.3.4.4 Evaluation of Maximum Zero-Load Communication Latency

Figure 4.10 and 4.11 show maximum zero-load communication latencies of all-to-all random traffic in a job, with different cache sizes, job sizes, and baseline switch latencies. Evaluation parameters, e.g. switching latency and cache miss penalty, are those of System-1 of the previous Section. The default entry size of the packet forwarding cache is 2,048.

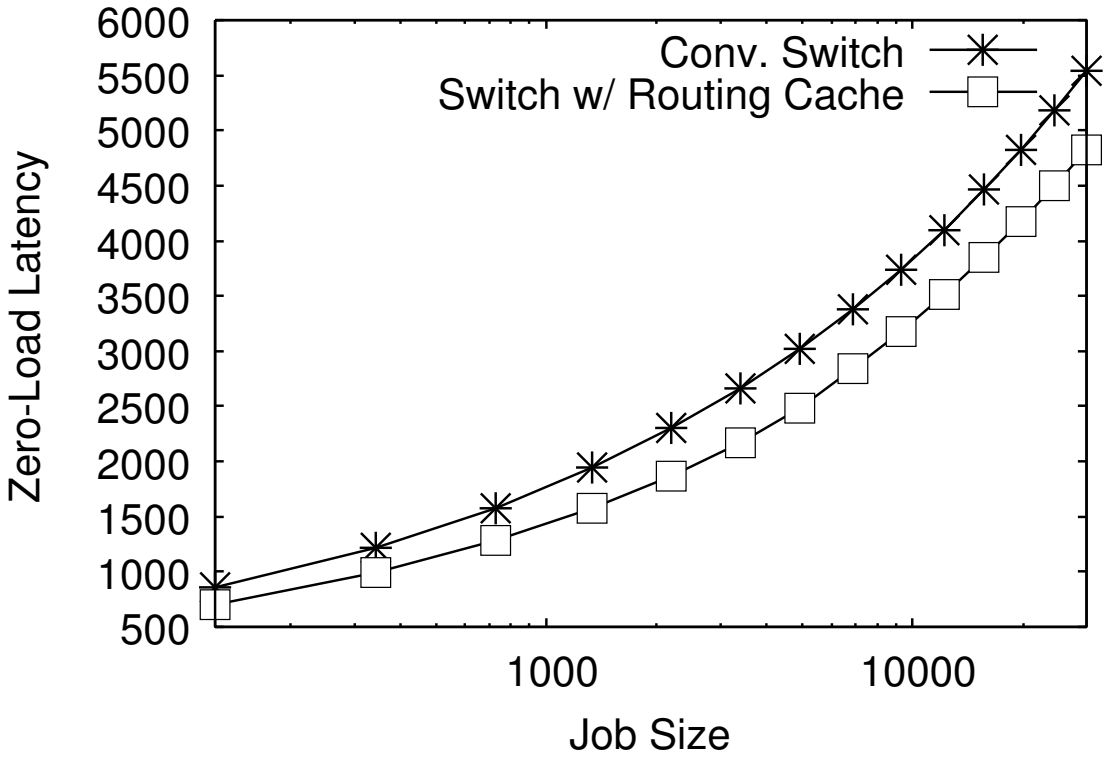


FIGURE 4.10: Zero-Load Communication Latency for Various Job Sizes.

Job Size Figure 4.10 shows maximum zero-load communication latencies along the sizes of jobs. As the job size becomes larger, capacity misses occur. The packet forwarding cache reduces the maximum zero-load communication latencies by 13% to 19%. Thus, the packet forwarding cache is effective to reduce the communication latency of a large scale job.

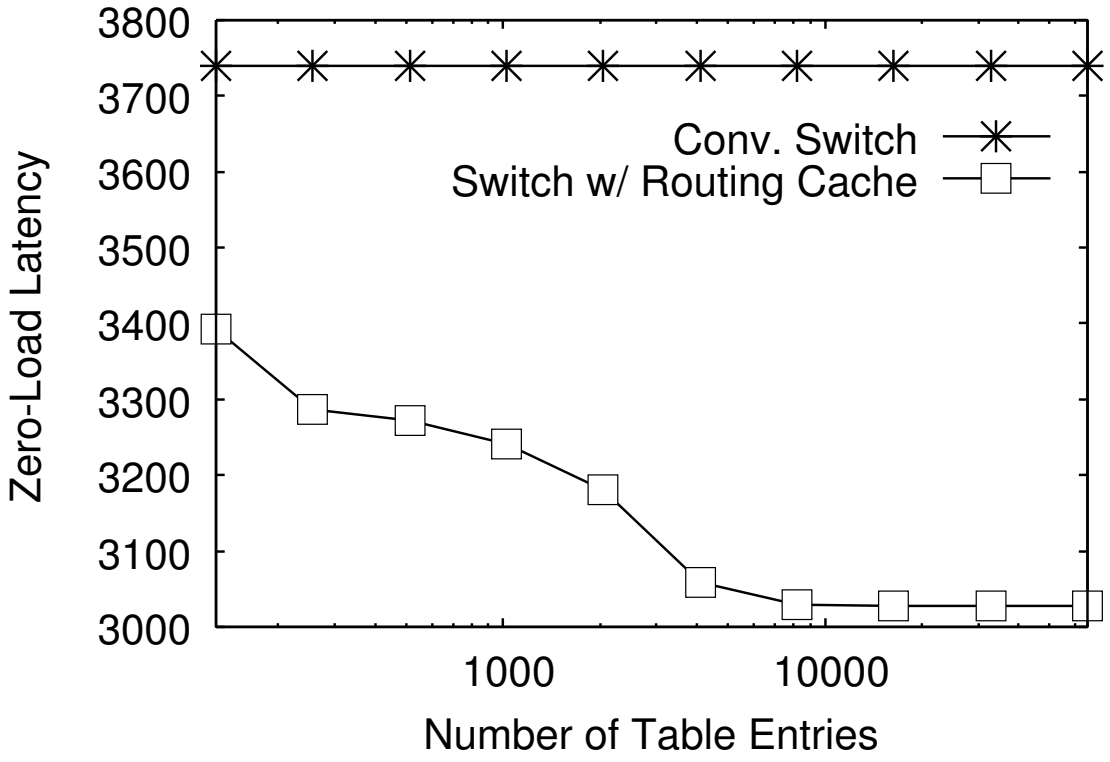


FIGURE 4.11: Zero-Load Communication Latency vs. Number of Cache Entries per Input Port.

Cache Entry Size Figure 4.11 shows maximum zero-load communication latencies along the sizes of the packet forwarding cache, in a large scale $21 \times 21 \times 21$ 3D Torus interconnection network. Maximum zero-load communication latencies of a conventional switch without the routing table cache (Conv. Switch) is also shown in the Figure. The maximum zero-load communication latency is reduced by 9% when the entry size of the packet forwarding cache is 128. Because the system has only 9,261($21 \times 21 \times 21$) computation nodes, the reduction of the maximum zero-load communication latency is no more than 19% when the entry size of the packet forwarding cache is more than 9,261.

4.4 Energy and Bandwidth

The energy consumption of CAM is high. TCAM(Ternary CAM) consumes 15 times more energy than SRAM (Static Random Access Memory) of the same capacity [96]. While there are researches on reducing static energy with power gating [97] or reducing dynamic energy with DVFS [98], the large energy consumption is still problematic.

4.4.1 Energy Consumption

Dynamic energy consumption on reading a CAM can be reduced because an access to the CAM is skipped when a cache access is a hit in a switch with the packet forwarding cache. In addition, most dynamic energy consumption on reading the CAM can be reduced when a 100% cache hit ratio is achieved in a switch with the packet forwarding cache. In this Section, effects of reducing energy consumption on looking up the CAM table on a switch with the packet forwarding cache is evaluated with a modeling.

The energy consumption S used in looking up the table of the switch is described with a model of energy consumption of an Internet router with a single routing table cache[67] below,

$$S = n(E_{cache} + E_{cam}r_{miss}) + (dP_{cache} + P_{cam}) \tag{4.5}$$

where E_{cache} and E_{cam} represents dynamic energy of a cache and a CAM, and P_{cache} and P_{cam} represents static energy of a cache and a CAM, respectively. Other parameters, n, d, and r_{miss} represent the number of packet processed in a second, switch degree, and cache miss rates, respectively.

Energy consumption of looking up a table of a conventional switch can be calculated by setting E_{cache} and P_{cache} to be 0, and r_{miss} to be 1, in Formula 4.5.

E_{cache} and P_{cache} is 0.039nJ and 13mW, respectively, by a CACTI6.5 [88] simulation. According to recent CAM energy consumption research[99, 100], E_{cam} and P_{cam} are set to 42nJ and 36mW, respectively. d is 7 ports, including one local port, assuming a 3D Torus topology.

Figure 4.12 shows S, which is energy consumption of looking up tables in switches, with the above parameters. In Figure 4.12, 0%, 50%(700Gbps), and 100%(1,400Gbps) are used for n for a 1400Gbps(200Gbps×7) link. Energy consumption of looking up tables is calculated for two cases, which use 1% and 0% as r_{miss}.

Figure 4.12 shows that the dynamic energy consumption of accessing the CAM is dominant on increasing the number of packet processing for a conventional switch. However, energy consumption of accessing the CAM of the switch with the routing table cache is largely reduced by reducing the number of accessing the CAM. While the static energy consumption of the packet forwarding caches for all input ports increases, it only shares 1% when the load is 50%. Hence, the reduction of dynamic energy consumption for accessing CAM is larger.

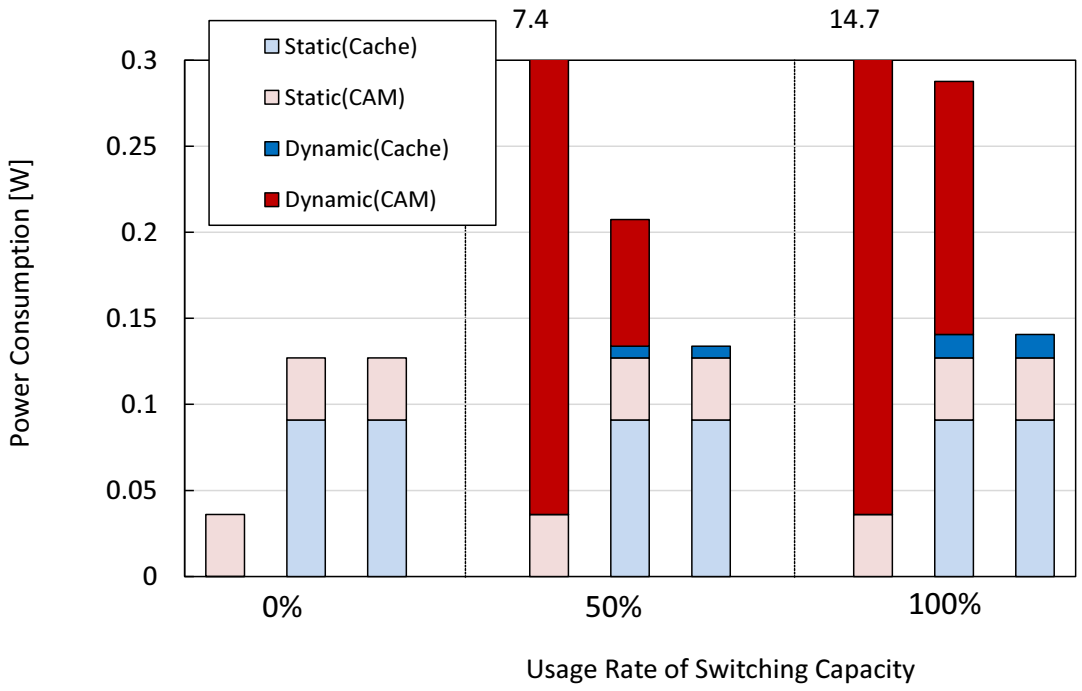


FIGURE 4.12: Power Consumption of Table Lookups in Three Cases of Forwarding Capacity Utilization.

Consequently, the energy consumption of a switch can be reduced by adopting the packet forwarding cache.

4.4.2 Forwarding Capacity

T_{base}, which represents a maximum (forwarding) throughput of a conventional switch, can be calculated with Formula 4.6, where L_{cam} represents a latency of accessing a CAM.

$$T_{base} = \frac{1}{L_{cam}} \tag{4.6}$$

Every packet needs one packet processing. A 25 nano seconds latency is needed to accessing a off chip CAM. As a result, T_{base}, which represents maximum throughput of a conventional switch with a single CAM, is limited to 40 Mpps (packet per second).

On the other hand, T_{rcache}, which represents maximum throughput of a switch with the packet forwarding cache, is calculated, where L_{cache} represents cache access latency, with Formula 4.7, because the CAM is accessed only when the cache misses.

$$T_{rcache} = \min\left(\frac{1}{L_{cache}}, \frac{1}{L_{cam} r_{miss}}\right) \tag{4.7}$$

The latency of routing processing on a cache miss is a sum of hash calculation, latency of accessing the cache, and latency of accessing the CAM. Thus, the latency on a cache miss is worse than that of a conventional switch. In contrast, the throughput is the same or higher than that of a conventional switch, because the maximum throughput of the switch with the packet forwarding cache is 40 Mpps, thanks to the pipelined stages even all packets cause cache misses.

The latency to accessing a 24 KiB packet forwarding cache is about 1 nano second, resulting in an 1 Gpps throughput per port. T_{rcache} of the switch with the packet forwarding cache is 1 Gpps, calculated with Formula 4.7 when almost 100% accesses are hits to the cache. This is 25 times higher than T_{base}, which is that of a conventional switch. The throughput decreases when the cache miss rate is more than 4%, with the CAM accessing being a bottleneck. However, T_{rcache} is 400 Mpps even the cache miss rate is 10%, that is 10 times higher throughput than T_{base} of a conventional switch.

Consequently, the throughput of routing processing increases much by adopting the routing table cache to a switch.

4.5 Summary

The performances of parallel applications are highly dependent on the communication throughput of interconnection networks between computation nodes. In this chapter, the application of packet forwarding cache to all input ports of switches is investigated to improve the switching latency, the line rate, and the performance of parallel applications. An incoming packet avoids large-latency accessing a CAM 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 × 84 Bytes × 8). The L1 cache size is strictly limited by the chip area. We thus focused on the case for up to 2K-node jobs.

In the interconnection network with 512 computation nodes, the conflict miss rate of a 2,048 entry 4-way set associative packet forwarding cache is less than 0.1%, with the cache simulation results, The performance of NAS parallel benchmark applications is

improved by 4.39x at maximum in a interconnection network with 256 computation nodes, with MPI simulation results of SimGrid v3.25.

Zero-load communication latency with the packet forwarding cache in a large scale interconnection network is evaluated. From the evaluation results, the reduction percentage of zero-load communication latency gradually decreases from 19% to 13% with the effects of capacity misses of the packet forwarding cache when used in a 9K computation node large scale interconnection network.

Additionally, with additional entries in the packet forwarding cache, the reduction percentage of zero-load communication latency gradually increases from 9% to 19%.

The adoption of the packet forwarding cache clearly improve the communication throughput of interconnection network and increase both the line rate and the performances of parallel applications Consequently, the packet forwarding cache is strongly recommended to be adopted in HPC switches.

Chapter 5

Switchable Node Reduction Function

In this Chapter, a switchable node reduction function for the packet forwarding cache is described. The switchable node reduction function enables to provides 100% cache hit rate except for the compulsory miss even for large scale jobs on typical network topologies.

5.1 Limitation of the Packet Forwarding Cache

The dissertation aims at enabling high-bandwidth low-latency interconnection networks for any job size. The analysis results in Chapter 4 indicate that $2K$ table entries are enough for almost 100% hit rates on the traffic within most jobs. However, the remaining elephant-nose (large) jobs, that would also be important in the HPC domain, suffer from the low cache hit rate on the cache switches.

A serious low cache hit rate is quantitatively illustrated in Figure 5.1 as the system size becomes large. Four-dimensional tori with a dimension-order routing and random traffic in which each node exchanges a message to a randomly selected destination node is assumed. Since parallel programs sometimes generate all-to-all communications, the random traffic should be considered, as described in Chapter 4. Cache misses are classified into compulsory, capacity, and conflict. As the job size increases, the capacity miss

becomes a crucial problem. Figure 5.1 illustrates the relationship between the job size and the capacity miss rates of the cache. An average high miss ratio close to 100% is suffered when the network size is large. This means that the effect of the cache becomes trivial for the traffic in the large job size.

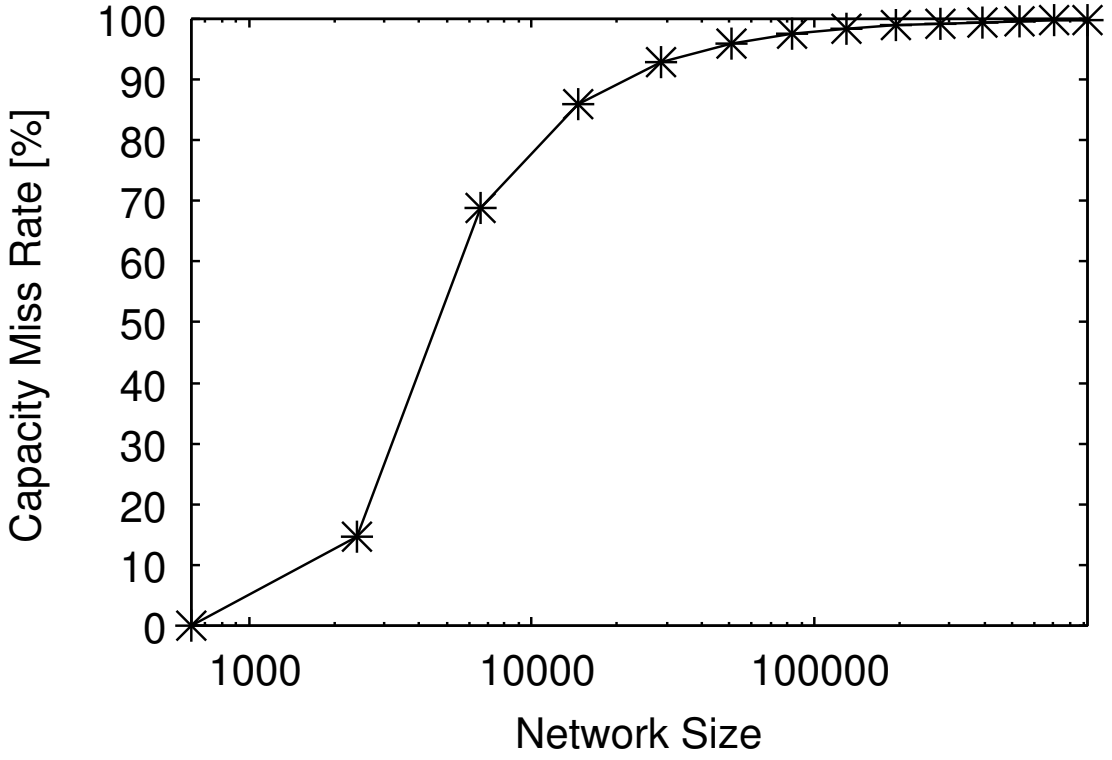


FIGURE 5.1: Capacity Miss Ratio on k-Ary 4-Tori (2K Cache Entries).

Figure 5.2 shows theoretical ideal bandwidths achieved from L1 caches(1ns, 0.4ns), TCAMs(20ns, 25ns), and the packet forwarding cache with a realistic miss rate, listed in Table 5.1. The x-axis represents the number of destination addressees, corresponding to the number of compute nodes. The y-axis represents the packet forwarding rate, corresponding to the maximum link bandwidth in Figure 5.2. Minimum size packets are assumed to arrive to each input port. The access latencies to L1 and L2 caches are obtained with CACTI v6.5. RC(L1-All-Hit-0.4ns) achieves an ideal maximum high bandwidth, assuming all accesses to the L1 cache are hits. TCAM(25ns) and TCAM(20ns) represents a current conventional interconnection network switch, where each can only achieves less than 100 Gbps bandwidth. RC(Real) decreases to less than 100 Gbps bandwidth when the number of compute node size is large, because the hit rates decrease to *zero*.

One may consider that a packet length may not always be small at an input port on a switch. Thus, the packet forwarding rate may not be a bandwidth bottleneck if a packet length is enough long. However, many-core processing generates a large number of tiny

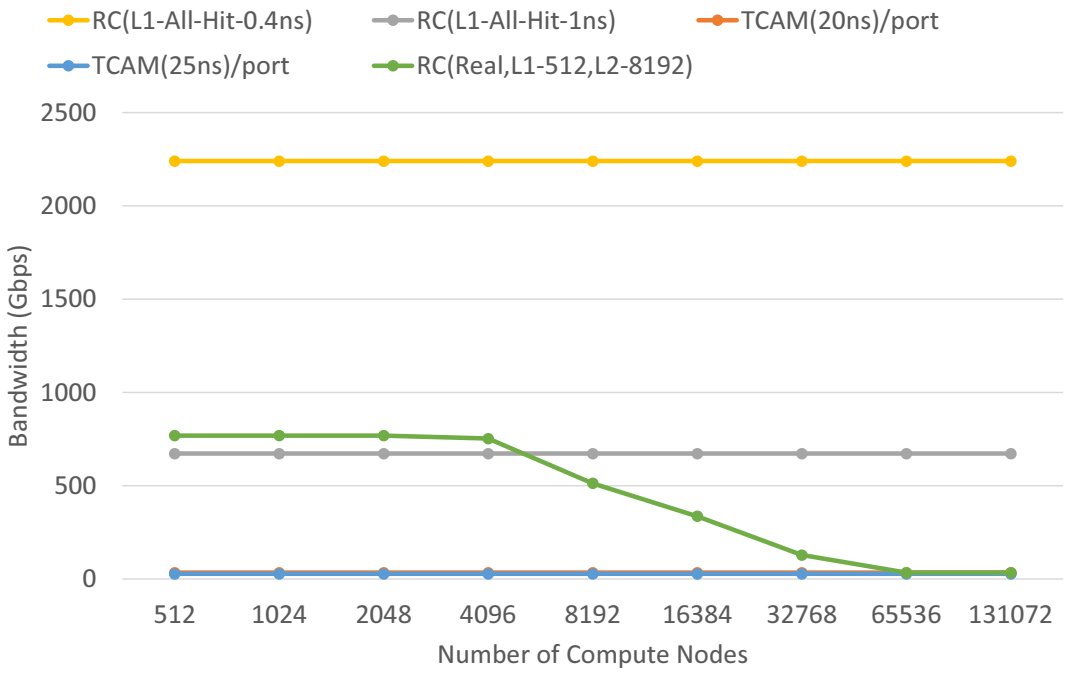


FIGURE 5.2: Number of Compute Nodes vs Link Bandwidth.

TABLE 5.1: Switch Parameters.

	L1	L2	TCAM
RC(L1-All-Hit-0.4ns)	0.4ns, 100%	-	-
RC(L1-All-Hit-1ns)	1.0ns, 100%	-	-
TCAM(20ns)	-	-	20ns
TCAM(25ns)	-	-	25ns
RC(Real)	0.4ns, 512 Entry	4.0ns, 8192 Entry	25ns

packets in parallel computing. Small messages (typically less than 3 KiB) constitute a major fraction of many HPC applications [10], and its high-throughput operation is listed up as a challenge to interconnection networks. As described in Section 1.2, the CAM internal latency does not decrease much year by year. The gap between a line rate and a packet forwarding rate will become a serious bottleneck in parallel computing. For example, it can provide 2.0 Tbps (=1/4ns × 1KiB × 8) for 1KiB packets. Link bandwidth will be 1.6 Tbps in the 2020s, as illustrated in Ethernet Roadmap 2020 [2], and over 2 Tbps link will be supported in the near future.

5.2 Switchable Node Reduction Function

5.2.1 Assumption and Behaviour

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

To achieve an 100% cache hit rate, the approach relies on a topology regularity. Existing parallel computers usually use k-ary n-cubes, fat tree, and Dragonfly network topologies, as described in Section 1.1. The approach targets to support an 100% hit rate only when they are used.

The main idea in this Chapter is to provide multiple node reduction functions that are optimized to the above common network topologies. The node reduction function assumes to use its custom node addresses. For example, the custom node address can be set as local identifier (LID) in InfiniBand. The procedure to use a switchable node reduction function is described as follows.

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

Figure 5.3 shows the block diagram of switches with the packet forwarding cache and the node reduction functions. When an incoming packet arrives, a cache tag corresponding to the destination address is obtained by the node reduction function. To easily maintain the consistency of CAM and cache, the switchable node reduction function outputs the index tag that is a key of routing cache.

Four datapaths on the node reduction functions in Figure 5.4, k-ary n-cube, fat-tree/Dragonfly, arbitrary topologies, and link aggregation (LAG) are prepared. The datapath for arbitrary topologies follows a CRC computation, and the CRC design follows [61].

The function to select a link among LAG is performed in parallel if LAG is used. The remaining two datapaths are stated as follows.

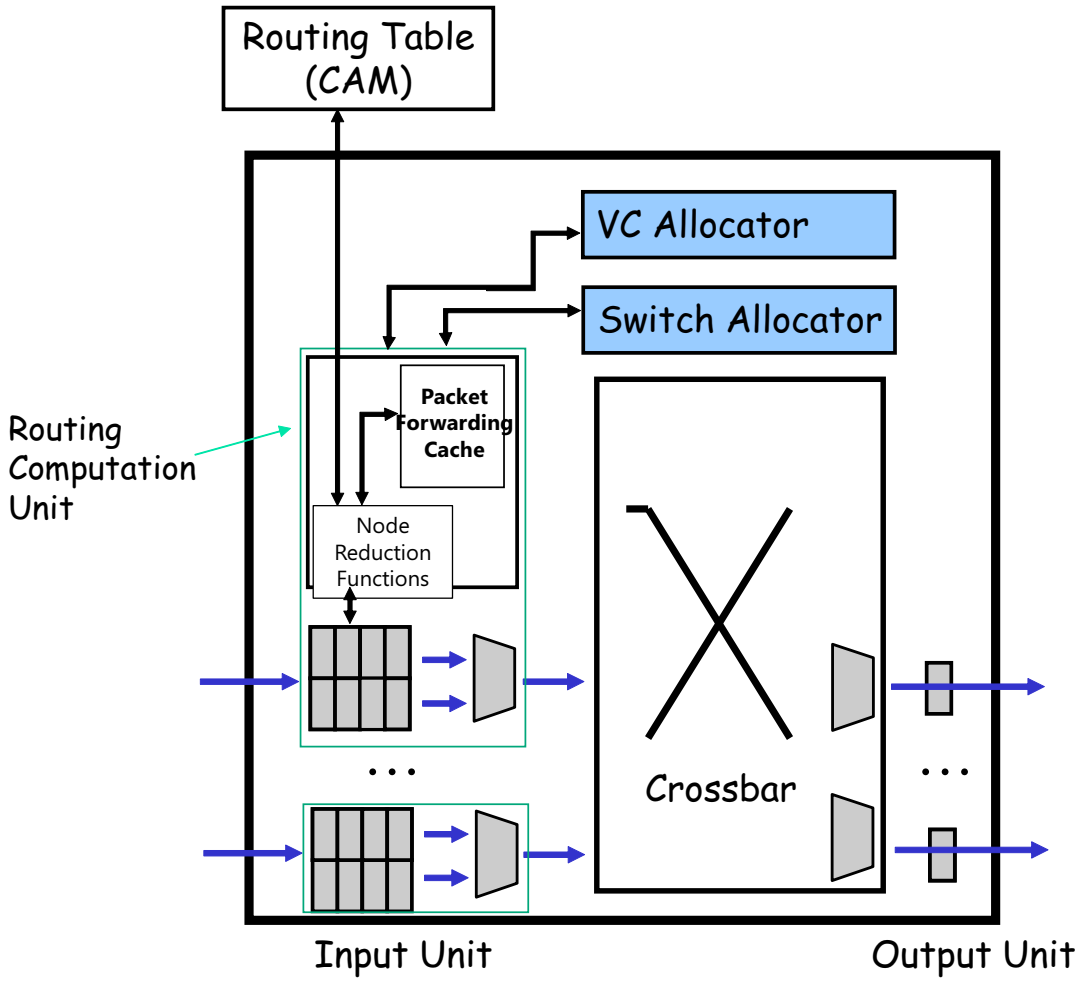


FIGURE 5.3: Block Diagram of Switch with Packet Forwarding Cache and Node Reduction Functions.

LAG is implemented logically the same as non-LAG networks, whereas a physical link within a LAG group is selected to use in transmitting each packet. A round robin, CRC of the target address, or simply the residual of the target address can be used. The CRC of the target address is used in Section 5.4.

5.2.2 General Node Reduction Function

In this section, a general node reduction function is described. It is intend to explain the general idea behind the node reduction functions instead of being actually implemented with hardware in a switch chip.

Figure 5.5 shows the overview of the idea behind the general node reduction function. A compute node has multiple output ports. Each port is connected to a set of compute nodes, e.g. three nodes set in Figure 5.5. All compute node has its own address. As a

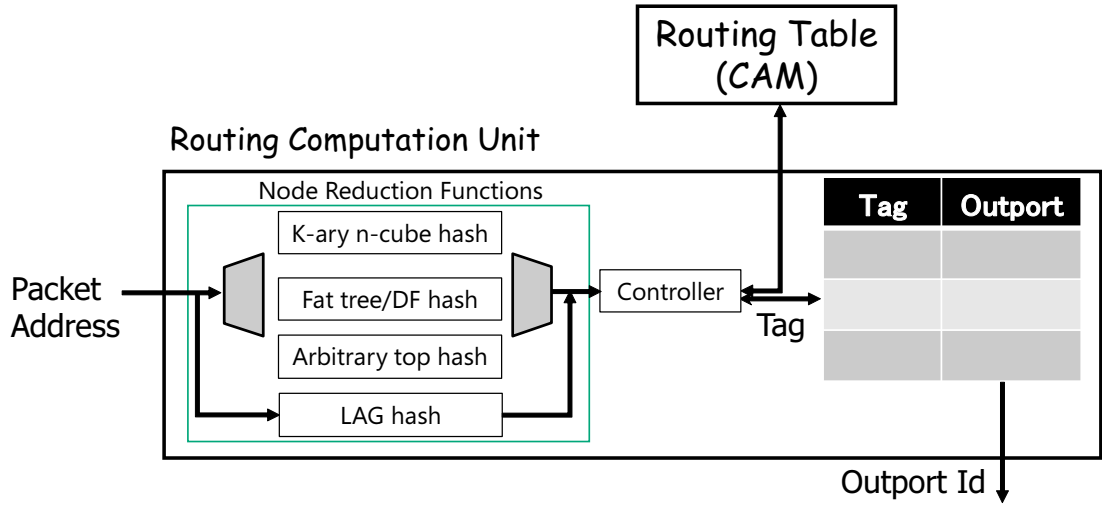


FIGURE 5.4: Routing Computation Unit and Node Reduction Functions.

result, a reduction function from addresses of a nodes set to a port number is able to be defined.

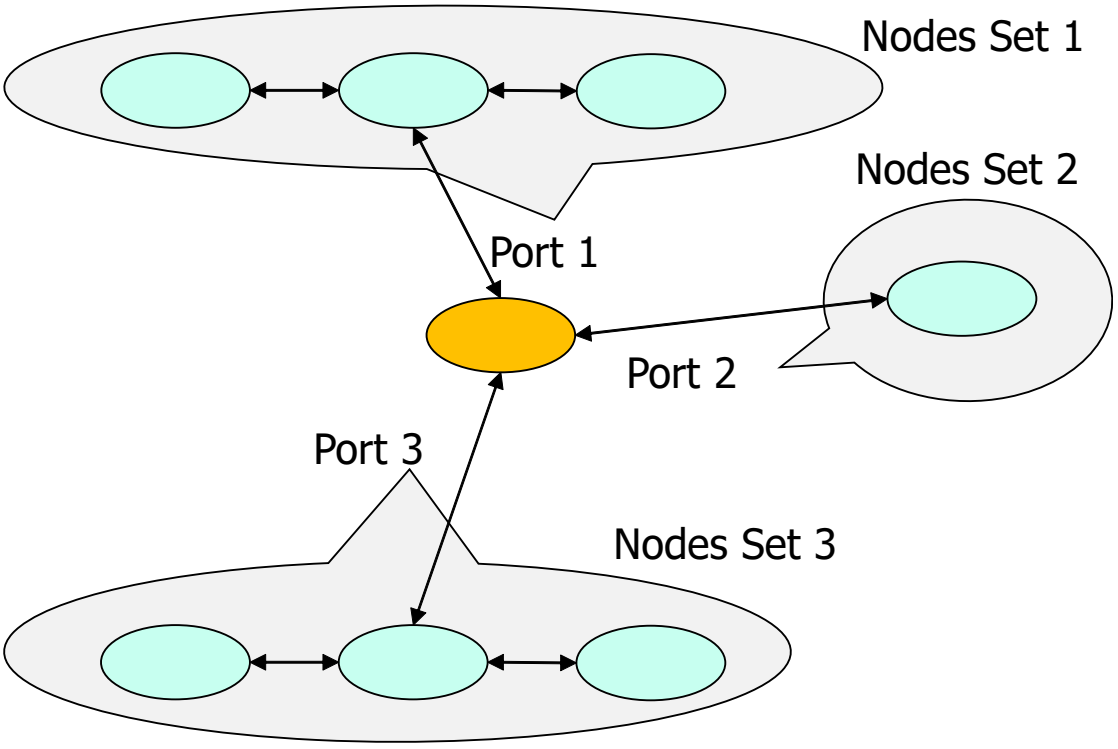


FIGURE 5.5: Overview of General Node Reduction.

Algorithm 1 shows the pseudo code of the general node reduction function. It obtains two addresses which represent the address of the current node address(c) and the address of the target destination address(d) respectively. The function body consists of a double loop, each iterates one of the two addresses, c and d.

Algorithm 1 General Node Reduction Function.

Require: Coordinates of current node $(c_0, \ldots, c_{n-1})$ and destination node $(d_0, \ldots, d_{n-1})$.
Ensure: Cache index *tag*.

```
1: procedure :
2:     for i = 0 to n − 1 do
3:         for j = 0 to n − 1 do
4:             if Handle(i, j, c_i, d_j) then
5:                 tag := getTag(i, j, c_i, d_j);
6:     tag := Local endpoint;
```

A *Handle* subfunction is used to determine whether the node reduction function terminates or not. If it does, another *getTag* subfunction is used to obtain the *tag*. If it does not terminate, the double for loop continues.

The key idea behind the switchable node reduction function is to create two hardware implementable subfunctions *Handle* and *getTag*, to achieve a 100% cache hit rate from the returned *tag* value. Because the *getTag* subfunction obtains (i, j, c_i, d_i) and returns a *tag*, it reduced the addressing space drastically, from the bitwidth of d_i to a simple *tag* value. As a result, the possible index space for the packet forwarding cache is dramatically reduced from $2^{bitwidth(d_i)}$ to only *tag*. Consequently, the packet forwarding cache with the switchable node reduction functions is able to achieve a 100% cache hit rate.

From the general node reduction function, special node reduction functions to handle typical network topologies can be derived, as described hereafter.

5.2.3 Typical Topologies Supported by Node Reduction Function

5.2.3.1 K-Ary N-Cube

A node reduction function to k-ary n-mesh/torus is stated in Algorithm 3. This can be derived from the general version of the function which is shown in Algorithm 1, where the "Handle" subfunction only handles the argument when $i = j$, and the subfunction "getTag" works as the pseudo code from line 3 to 7 of Algorithm 3.

Dimension-order routing is assumed to use in the network. Each compute node is assigned to the n-dimensional coordinates $(X_{n-1}, \ldots, X_1, X_0)$, and each switch has compute nodes. Each output-port information in the cache corresponds to a link.

Firstly, the case for k-ary n-meshes is considered. The required number of table entries is minimized, that is the number of output ports on a switch. Figure 5.6 shows an example of mesh network. 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

Algorithm 2 Node reduction function on k-ary n-mesh.

Require: Coordinates of current node $(c_0, \ldots, c_{n-1})$ and destination node $(d_0, \ldots, d_{n-1})$.
Ensure: Cache index *tag*.

```
1: procedure :
2:     for i = 0 to n − 1 do
3:         X_offset := d_i − c_i;
4:         if x_offset > 0 then
5:             tag := X_{i,+};
6:         else if x_offset < 0 then
7:             tag := X_{i,−};
8:     tag := Local endpoint;
```

same table entry ($X_{0,+}$) at the switch (1,1) in Figure 5.6. In the Figure, each output port is marked as $X_{0/1,+/-}$, that is the table entry.

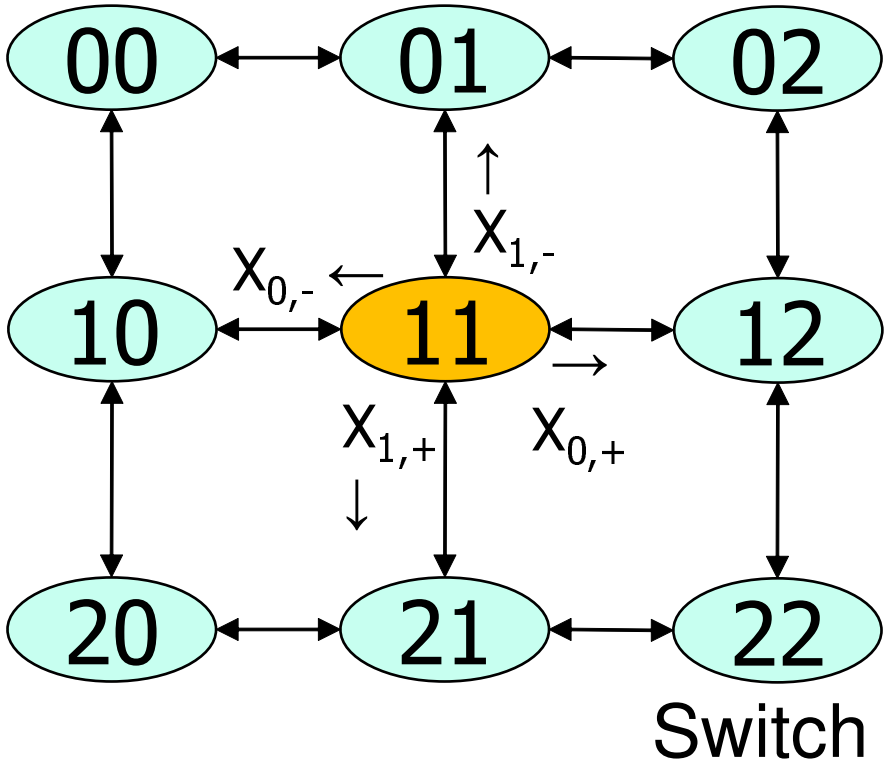


FIGURE 5.6: An Example of 3-Ary 2-Mesh.

The reduction function of node "11" works for any target nodes in the network. Without the reduction function, the possible target addresses are all nine network nodes, including the "11" node itself, which is usually called as a sink port.

With the reduction function, the possible number of the target addresses are reduced to five, which is the sum of the number of the output ports and one sink port, as shown in Figure 5.7.

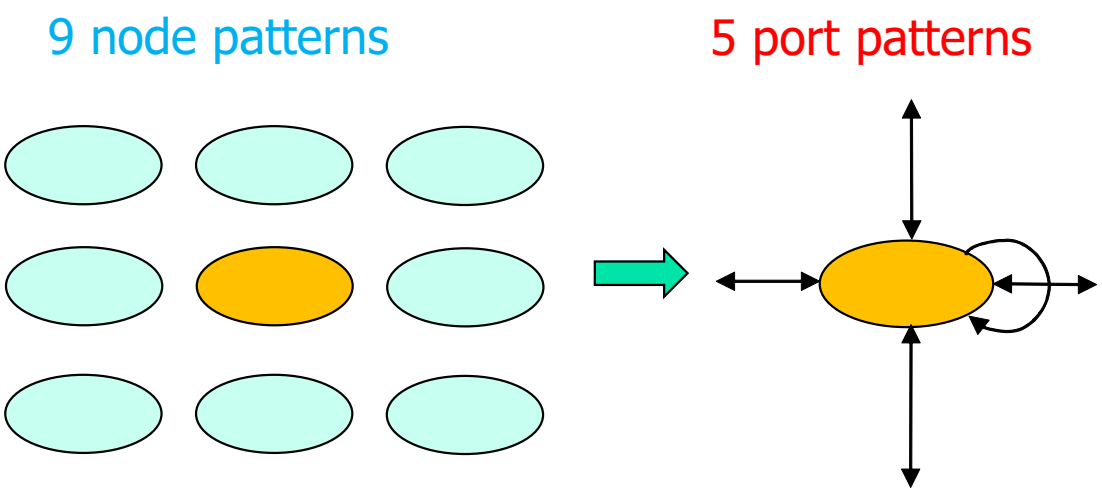


FIGURE 5.7: Reduction of Routing Patterns on 3-Ary 2-Mesh.

This characteristic is the key of the reduction functions, which can reduce the number of possible target addresses from the number of all network nodes to the number of output ports. This reduction can be significant when the network is huge, because the number of output ports is no more than a few hundreds while the number of nodes can be more than a hundreds of thousands.

The regularity of dimension-order routing on k-ary n-cubes is fully utilized to minimize the number of table entries. Indeed, Figure 5.7 represents that the number of table entries can be classified into five from nine destinations. Thus, only five table entries are needed for k-ary 2-meshes. More generally, k-ary n-meshes require only $2n+1$ table entries.

Figure 5.8 shows the address format for the k-ary n-cube mesh/torus network. The address is divided to n address chunks. Each chunk is corresponding to a dimension of the network. In the figure, the LSB chunk represents the first dimension of the mesh network. The neighboring chunk of the LSB chunk represents the second dimension. Consequently, N dimensions can be represented with the N chunks.

Figure 5.9 shows the format of the address chunk. Each chunk consists of K bits, representing the address of a node in a dimension of the network. With the K bits, $2^k - 1$ nodes can be addressed, representing the node in the dimension.

The address format is assumed to be 24 bits. The chunk size can be 24 bits address format is assumed. K can be 2, 3, and 4, representing 2- to 16-ary mesh/torus topologies. Consequently, the maximum number of chunks in an address is $12 = 24/2$.

Figure 5.10 shows an implementation of a node reduction function that supports mesh topologies. It consists of a splitter, 12 comparators, and a multiplexer, following with

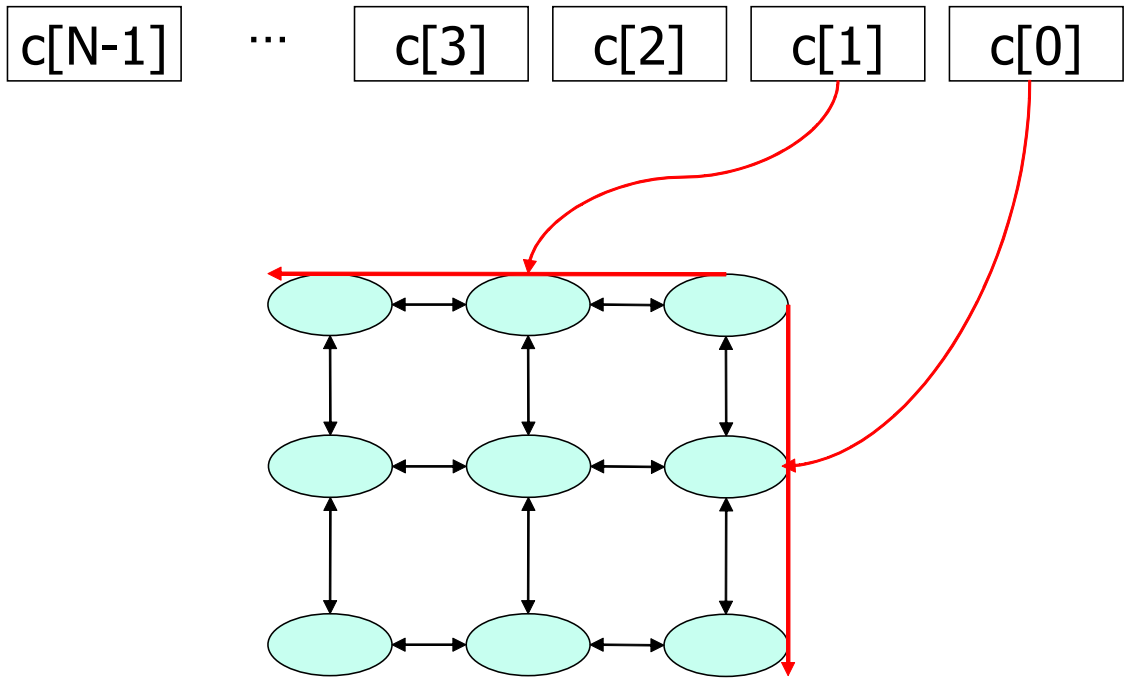


FIGURE 5.8: K-ary, N-cube Mesh/Torus Address Format.

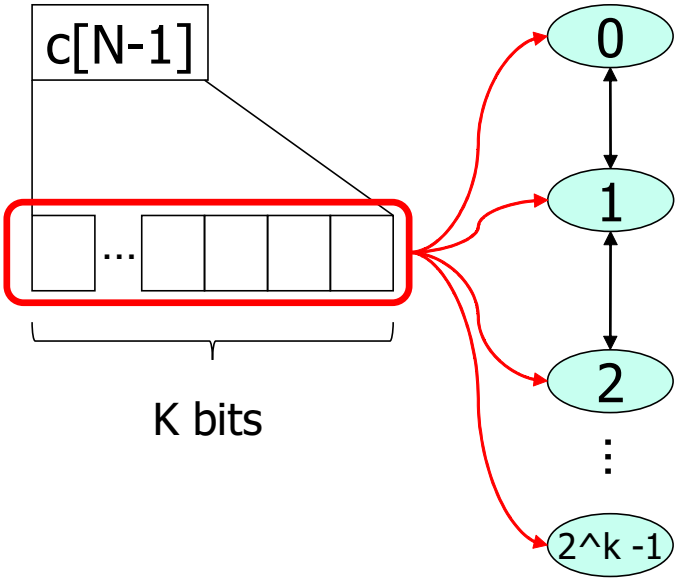


FIGURE 5.9: K-ary, N-cube Mesh/Torus Address Chunk.

the Routing Cache, to realize the functionality of Algorithm 2. Note that the "for" loop in Algorithm 2 is implemented with the 12 comparators, because of the maximum number of chunks is 12. The comparators can be accessed in parallel, resulting in a shorter access latency than using a state machine.

Secondly, we consider the case for k-ary n-tori. On tori, there are the wraparound channels, as shown in Figure 5.11. It makes the node reduction function, which is shown

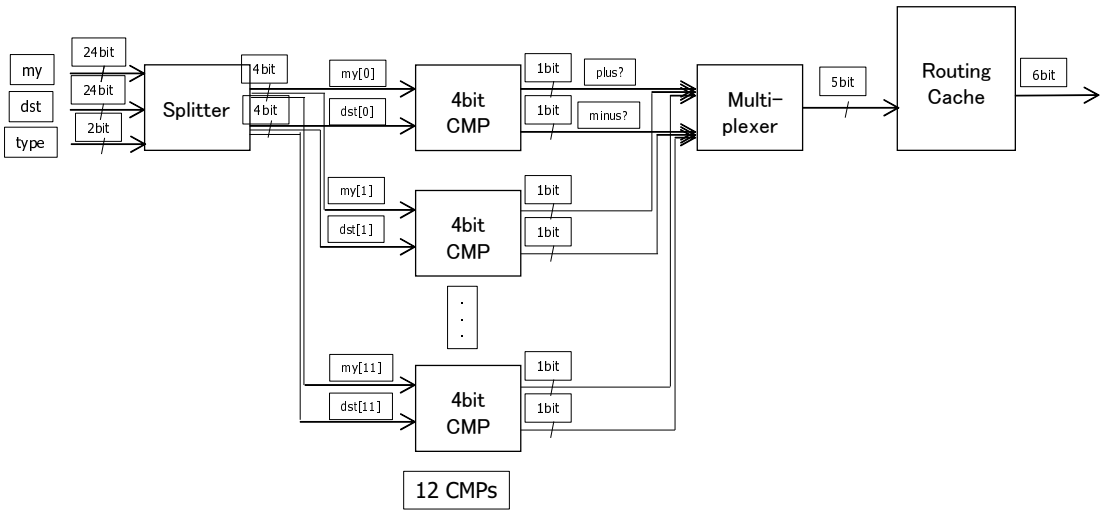


FIGURE 5.10: Implementation of the Mesh Node Reduction Function.

Algorithm 3 Node reduction function on k-ary n-torus.

Require: Coordinates of current switch node $(c_{n-1}, \ldots, c_0)$ and destination compute node $(d_{n-1}, \ldots, d_0)$.
Ensure: 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,−};
6:          else if x_offset > 0 then
7:              tag := X_{i,+};
8:          else if x_offset < −⌈k/2⌉ then
9:              tag := X_{i,+};
10:         else if x_offset < 0 then
11:             tag := X_{i,−};
12:     tag := Local ComputeNode;
```

in Algorithm 3, different from that in k-ary n-mesh.

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 it is not shown in Algorithm 3. Each table entry in Algorithm 3 should be stored so as to avoid the conflict miss on n-way associative cache. Conflict misses can be avoided by using the CRC number of the target addresses, or round-robin of the target address or the target port number itself.

Node reduction function on k-ary n-torus with supporting LAG is shown in Algorithm 4. In the Algorithm, each output logical port is equipped with lag_num physical output ports.

Figure 5.12 shows a logical configuration of a network switch with LAG. Each switch has four logical ports, $port\_minus[0, 1]$ and $port\_plus[0, 1]$. Each logical ports consists of P

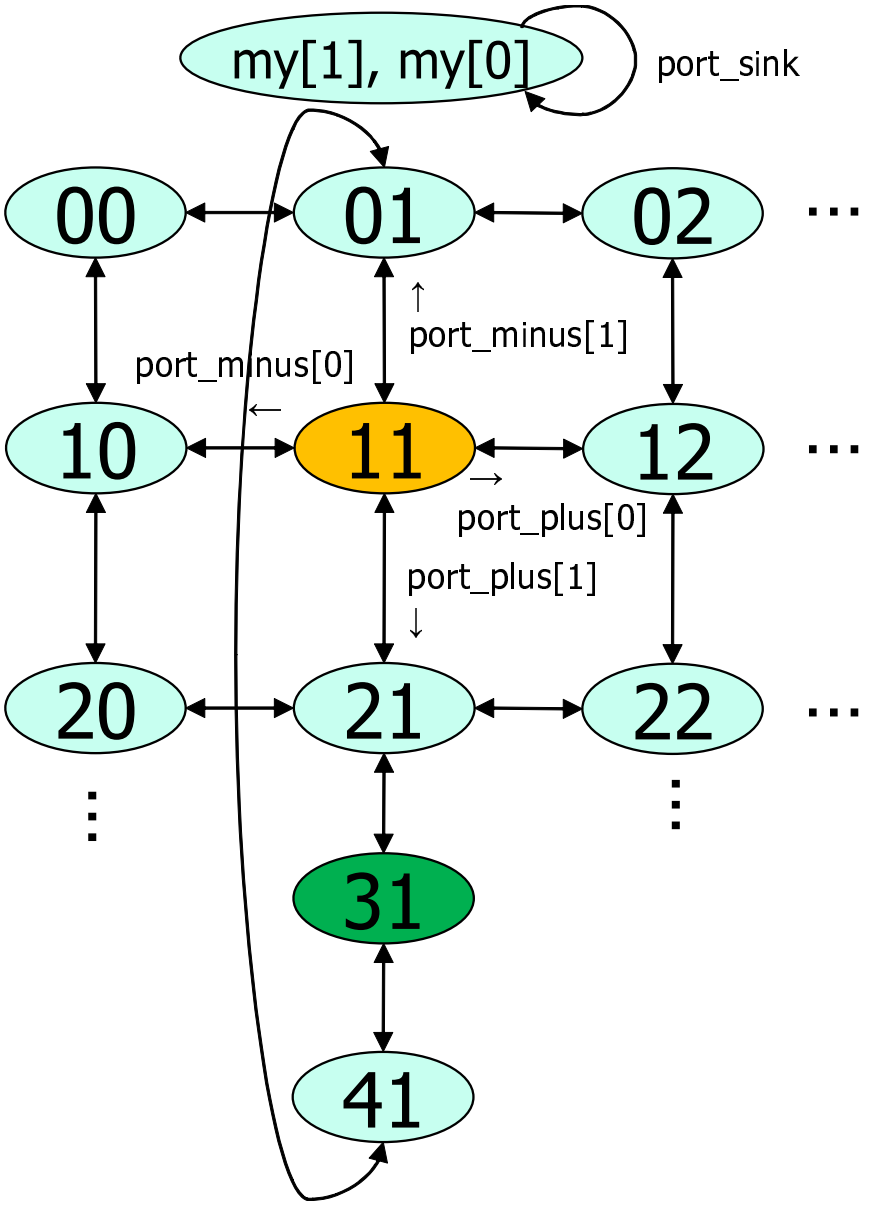


FIGURE 5.11: Reduction of Routing Patterns on 5-Ary 2-Torus.

physical ports, resulting in a LAG logical port. Each physical port is selected according to a CRC number of the target address. Consequently, an aggregated bandwidth is achieved when a large number of packets are transmitted with the LAG port.

Figures 5.13 shows an example of torus topology routing, equipped with LAG ports. Note that the direction of packet transmitting in a dimension is determined by the relative position of the source node and the target node, i.e. the node "41" should be routed to the upper direction from the node "11", while the routing is not feasible in the mesh network. *c_be* is calculated and used to determine the direction where a packet should be routed along the dimension. In the figure, the node "31" is the *c_be* node of the node "11" in the first dimension, which can be calculated as $3 = (1 + \lfloor 5/2 \rfloor) \mod 5$.

Algorithm 4 Node reduction function on k-ary n-torus with supporting LAG.

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

procedure :
  $lagID = d \mod lag\_num$;
  for $i = 0$ to $n - 1$ **do**
    $X_{offset} := d_i - c_i$;
    if $x_{offset} > \lceil \frac{k}{2} \rceil$ **then**
      $tag := X_{i,-[lagID]}$;
    else if $x_{offset} > 0$ **then**
      $tag := X_{i,+[lagID]}$;
    else if $x_{offset} < -\lceil \frac{k}{2} \rceil$ **then**
      $tag := X_{i,+[lagID]}$;
    else if $x_{offset} < 0$ **then**
      $tag := X_{i,-[lagID]}$;
  $tag := Local\ ComputeNode$;

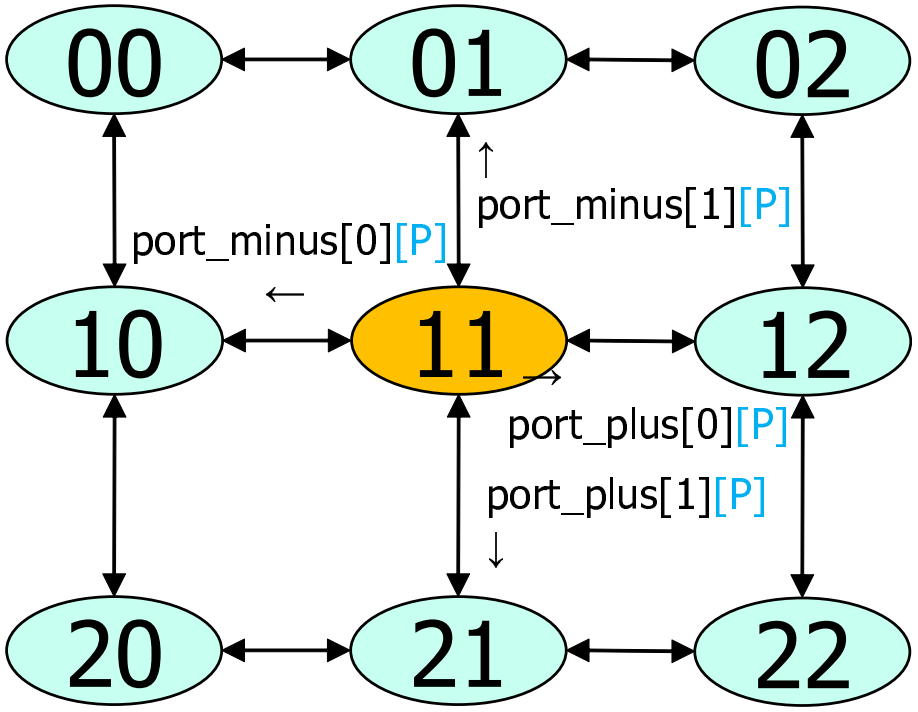


FIGURE 5.12: Aggregated P Ports for All Logical Port of Node "11" in a Mesh Network.

5.2.3.2 Fat Tree

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

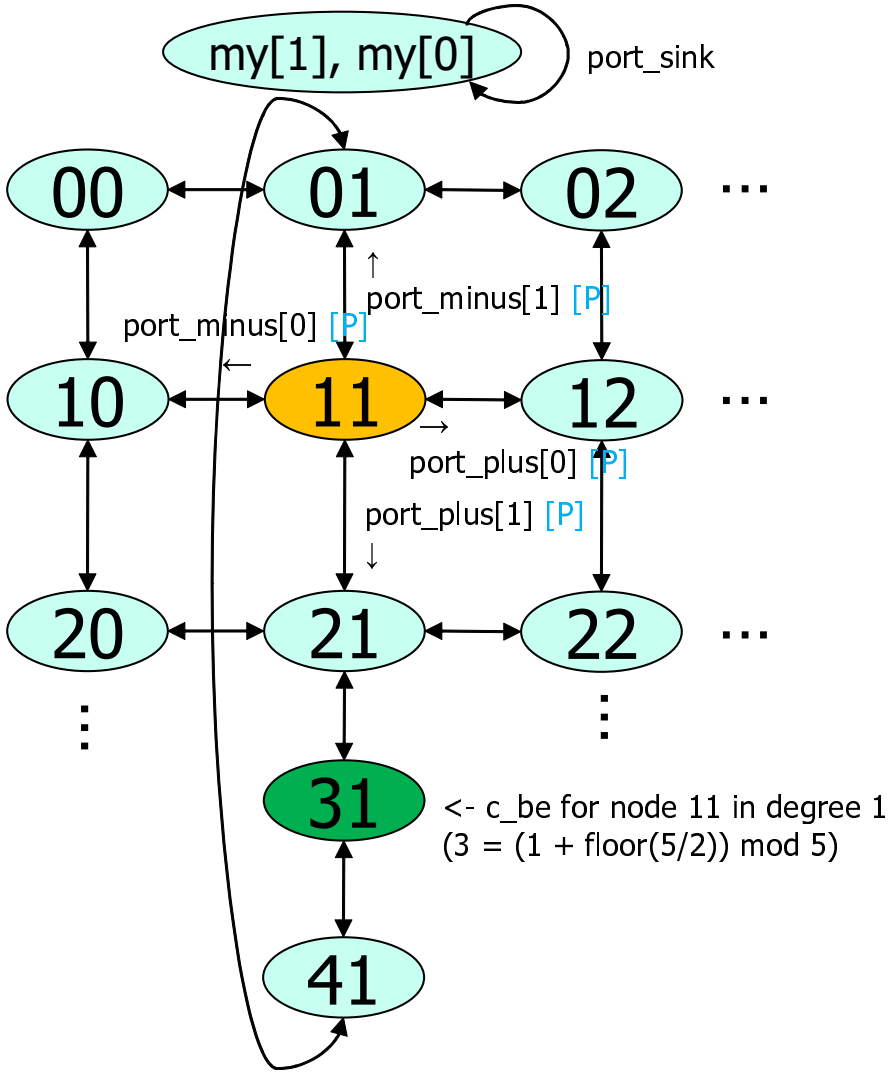


FIGURE 5.13: Aggregated P Ports for All Logical Port of Node "11" in a Torus Network.

Algorithm 5 Node Reduction Function on n-Dimensional Fat Tree.

Require: Coordinates of current switch node $(dim, c_{n-1}, \ldots, c_0)$ and destination compute node $(d_n, \ldots, d_0)$.
Ensure: 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:     tag := down_{d_i};
```

Figure 5.15 shows the node address format for Fat tree topology. The address consists of $D + 1$ chunks, where D denotes the number of dimensions in the Fat tree network, four in the figure as an example.

Each node address chunk format is shown in Figure 5.16. Each chunk consists of K bits, where K denotes the logarithmic of the number of the switches in the same dimension of the Fat tree network.

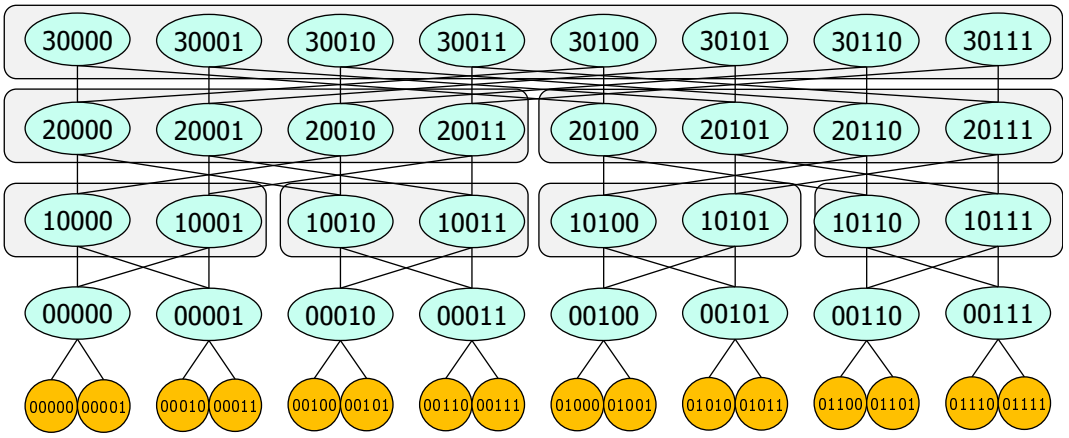


FIGURE 5.14: An Example of 2-Ary 4-D Fat Tree. (n=4, k=2)

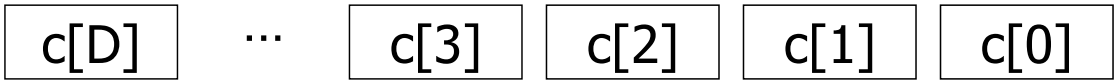


FIGURE 5.15: D-dimensional K-ary Fat-tree Node Address Format.

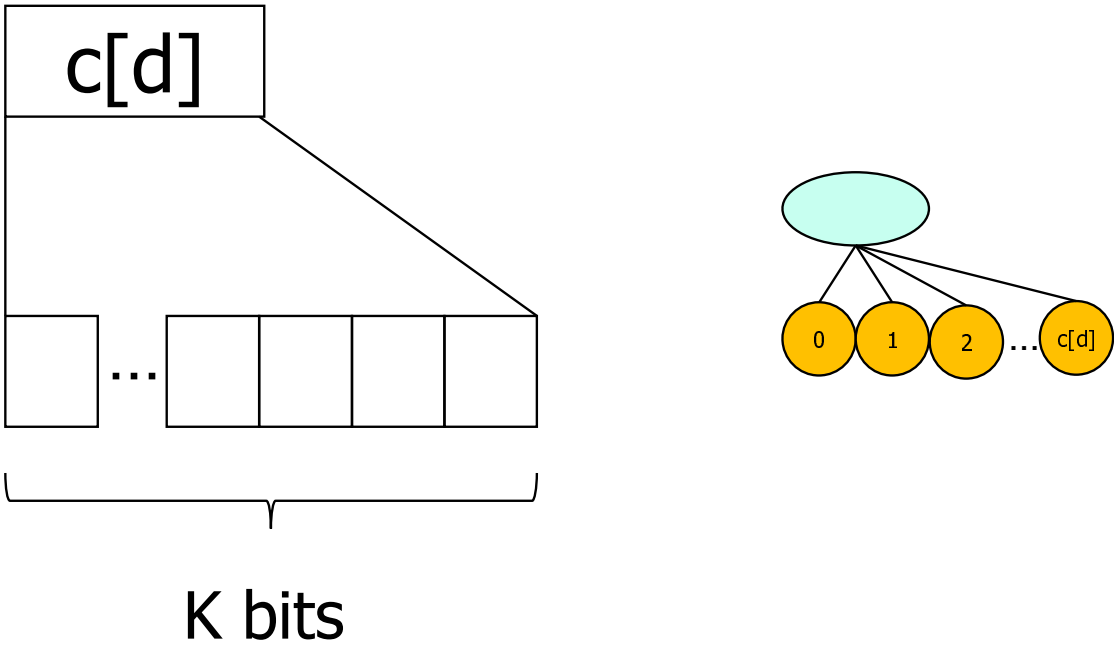


FIGURE 5.16: D-dimensional K-ary Fat-tree Node Address Chunk Format.

Figure 5.17 shows the switch address format for Fat tree topology. The address consists of a d with D chunks, where D denotes the number of dimensions in the Fat tree network, four in the figure as an example. d denotes the number of the dimension in the Fat tree network.

Each switch address chunk format is shown in Figure 5.18. Each chunk consists of

K bits, where K denotes the logarithmic of the number of the switches in the same dimension of the Fat tree network.

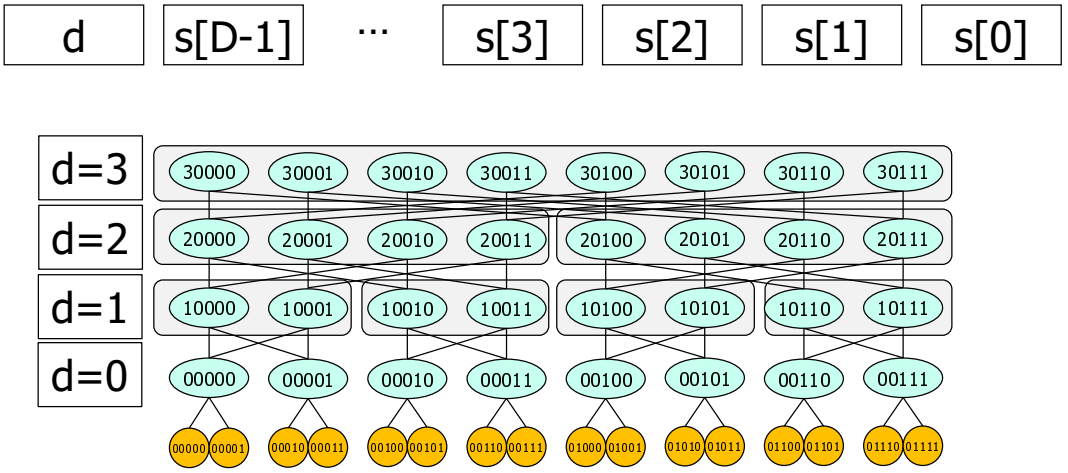


FIGURE 5.17: D-dimensional K-ary Fat-tree Switch Address Format.

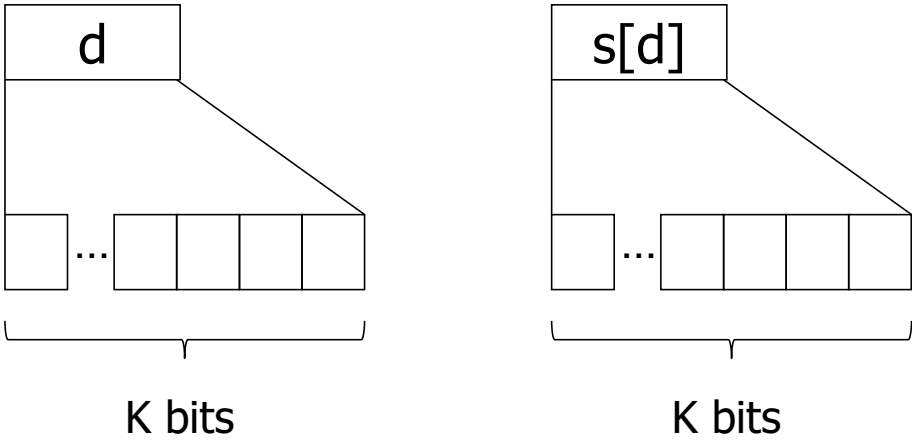


FIGURE 5.18: D-dimensional K-ary Fat-tree Switch Address Chunk Format.

Algorithm 5 represents the node reduction 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 $c_j\{0 \leq j < n, 0 \leq c_j < k\}$ 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 $0 \leq j < k$. The links up_0 and $down_0$ are connected to the left-most switch of neighboring up/down layers.

Figure 5.14 represents an example of the node addresses in Algorithm 5. In the Algorithm, the matching to a dimension is represented in Figure 5.19. In the reduction function, the top d chunks of the node address are compared to the top d chunks. If

they do not match, the routes are up, p_u[] in the figure. If all the comparison matches, the routes are down, p_d[] in the figure.

Algorithm 5 is also a derived function of the general Algorithm 1. In this case, the "Handle" subfunction only handles once, e.g. $i = j = 0$, and the subfunction "getTag" works as the pseudo code from line 2 to 4 of Algorithm 5.

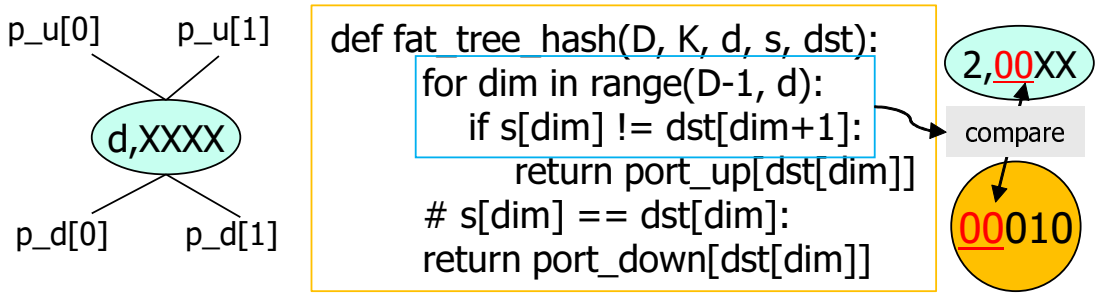


FIGURE 5.19: Comparisons of Fat Tree Hardware Reduction Function.

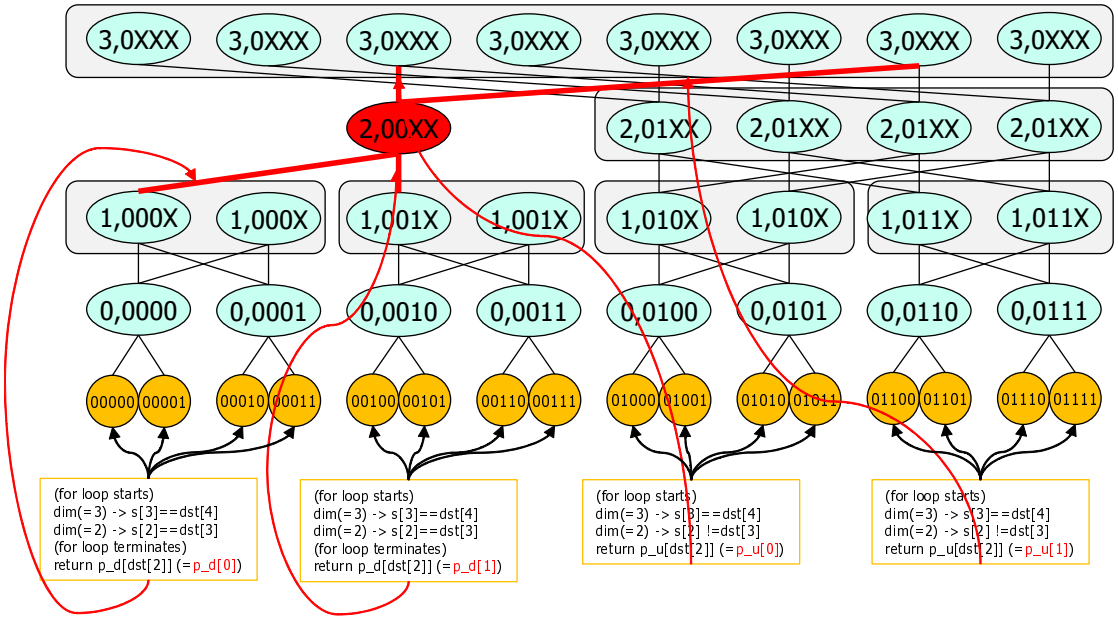


FIGURE 5.20: Packets Routing in a Fat Tree Network.

5.2.3.3 Dragonfly Network Topology

The Dragonfly is a meta network 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 node reduction configuration of fat tree in Algorithm 5. We illustrate the Dragonfly in Figure 5.21 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 5.21. 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 node reduction function of fat tree also works for each switch node.

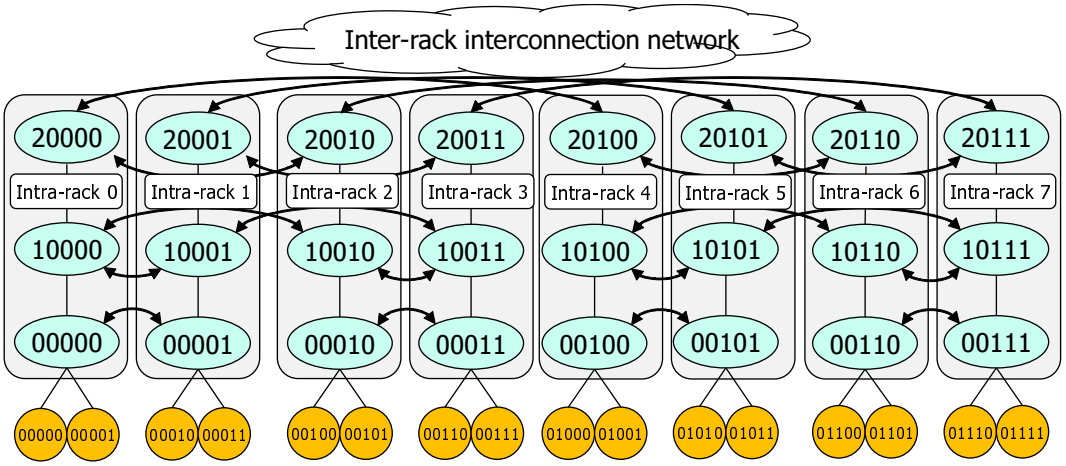
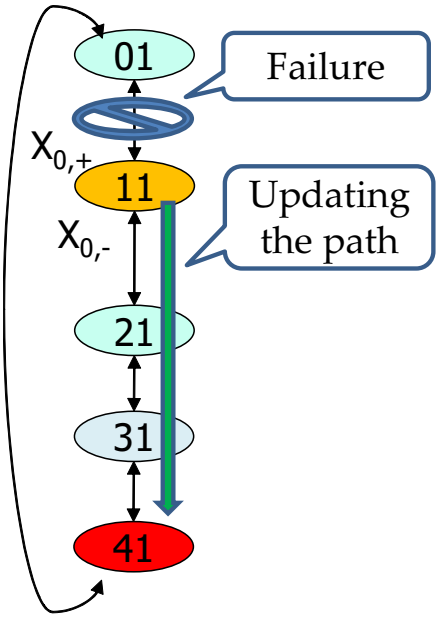


FIGURE 5.21: Dragonfly Corresponding to the Fat Tree in Figure 5.14.

5.3 Limitation of the Switchable Node Reduction Function

A same node reduction function can be shared by tori and meshes. It can support some node reduction function 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 5.22.

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 node reduction 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.



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 5.22: An Example of a Faulty 5-Ary 1-Cube.

5.4 Evaluation of the Switchable Node Reduction Function

5.4.1 Condition

The switchable node reduction function is designed with the Nangate 45nm Open Cell Library [101]. 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 [61]. It is essentially needed even if the switchable node reduction 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 switch 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 node reduction 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 over subscription 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 node reduction function to support $1:1$ over subscription 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.

5.4.2 Results

5.4.2.1 Cache Approach

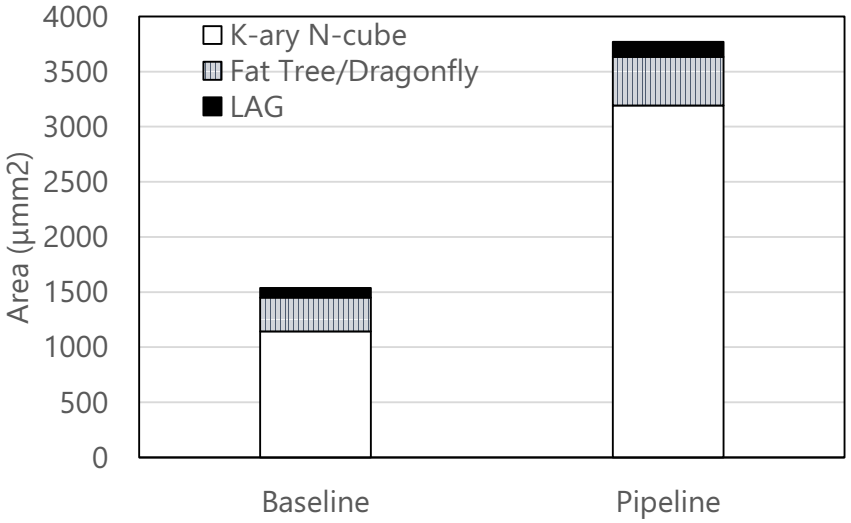


FIGURE 5.23: Node Reduction Function Area Overhead at an Input Port.

Figures 5.23 and 5.24 illustrate the area and latency overheads of the switchable node reduction function, respectively. We 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 5.4 for the link aggregation.

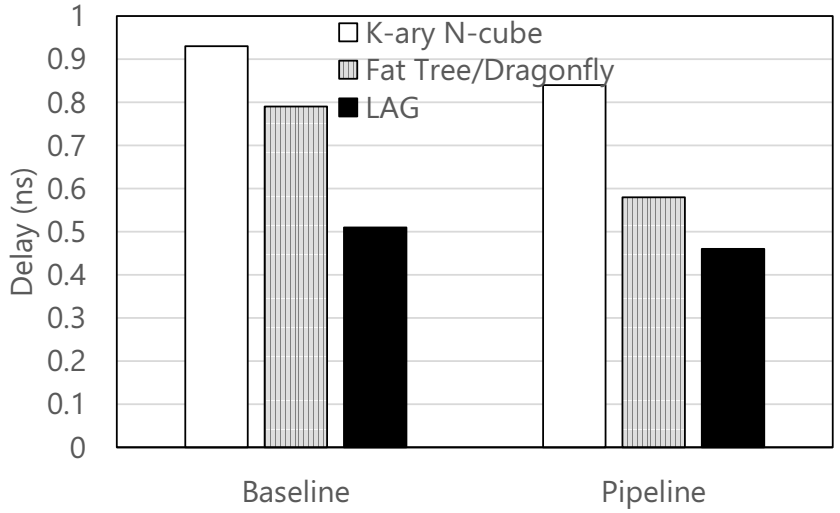


FIGURE 5.24: Node Reduction Function Latency Overhead at an Input Port.

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 μmm^2 in the baseline and 352 μmm^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 μmm^2 and 165 μmm^2 on the baseline and the pipeline, respectively. The operation of the node reduction function for fat trees is based on a bit-wise comparator, and it becomes lightweight compared to that for k-ary n-cubes. Thus, The node reduction function for k-ary n-cubes occupied 85% of the total amount of the hardware for the switchable node reduction function.

Although there is a trade-off between generality and area overhead, its area overhead is small compared to a switch ASIC, e.g., 329 mm^2 of ICC chip in the K computer (10 ports, Fujitsu Semiconductor 65 nm process technology). In this context, we conclude that the area overhead of each node reduction 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 node reduction function and LAG computational function are done in parallel. Thus, the latency can be regarded as the maximum value of the switchable node reduction function and LAG computational function. It is less than 1 nano second. If we support the above limited configuration, the latency further decreases. For example, when only 6-cubes are supported, the latency becomes 0.54 ns and 0.47 ns on the baseline and the pipeline, respectively. If three-stage pipeline is used, each stage can be completed within most 0.72 ns. The access is fully pipelined with three stages, the packet forwarding throughput becomes 933 Gbps (1/0.72 ns $\times$ 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.

5.4.3 Parallel Application Performance

5.4.3.1 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 (v3.25) [90] to simulate the execution of unmodified parallel applications that use the Message Passing Interface (MPI) [102]. The parameters are listed in Table 5.2.

Table 5.2: Parameters of SimGrid Simulation.

	Torus	Dragonfly
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	100 GFlops / core	100 GFlops / core
Network topology	4-D torus	Dragonfly
Number of Nodes	32 / 256 switches	32 / 256 Switches
	$2^3 * 4$ / 4^4	4 Groups × 8 nodes / 16 Groups × 16 nodes

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

A conventional switch that uses CAM assumes to use a 200 Gbps, 400 Gbps, 800 Gbps, and 1.6 Tbps link. 1.6 Tbps is used for simulating link aggregation. It would not guarantee 400 Gbps 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 [84] (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 [103].

5.4.3.2 Simulation Results

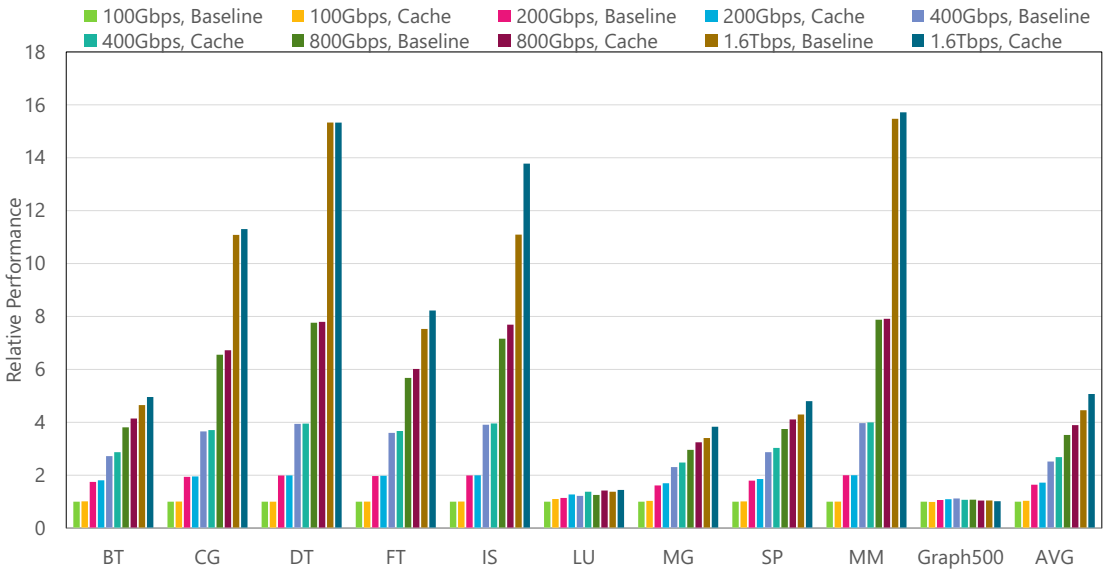


FIGURE 5.25: Relative Application Performance (256 Nodes, 4-D Torus).

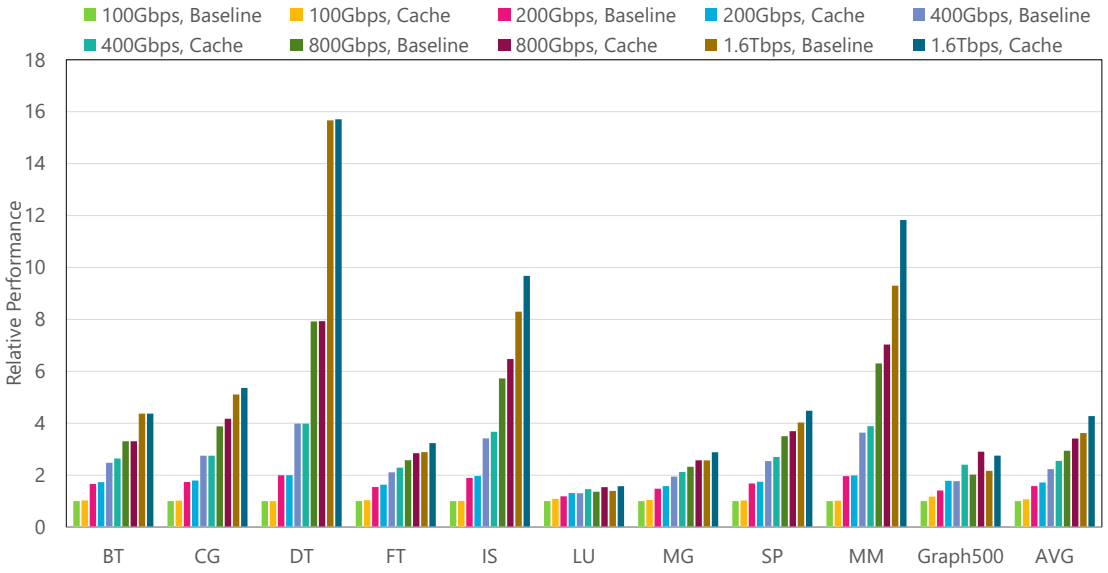


FIGURE 5.26: Relative Application Performance (256 Nodes, Dragonfly).

Figures 5.25 and 5.26 show performance results for the baseline and the cache switches, normalized to the performance achieved by the baseline switch using 100 Gbps links with 256 computation nodes. Figures 5.27 and 5.28 show the relative performance results of 32 computation nodes. 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 51 ns, while the conventional switch takes a 60 ns delay. The delay difference results in the performance improvement by the cache switch. The

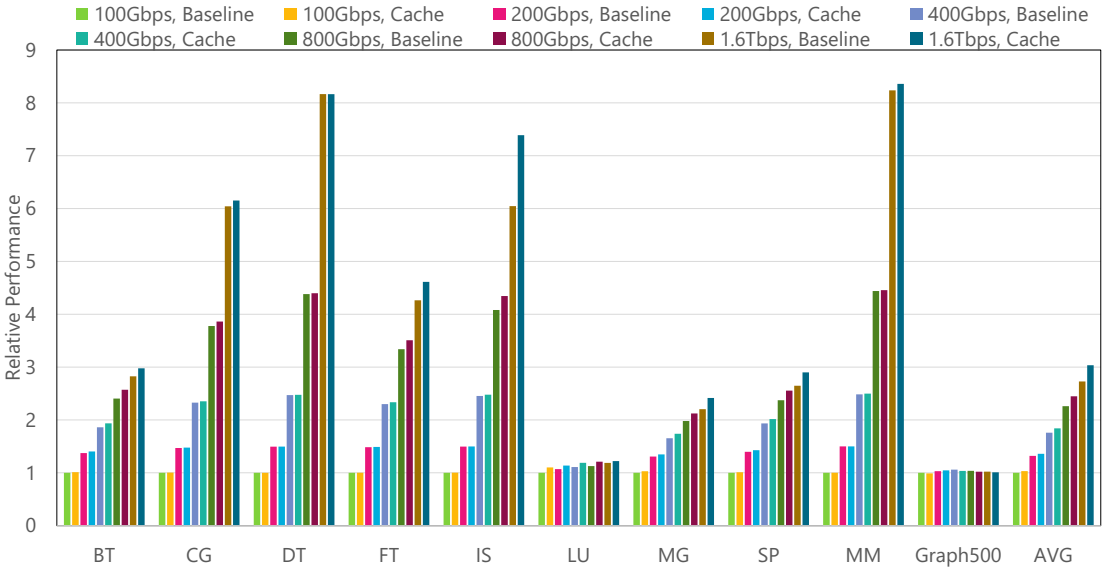


FIGURE 5.27: Relative Application Performance (32 Nodes, 4-D Torus).

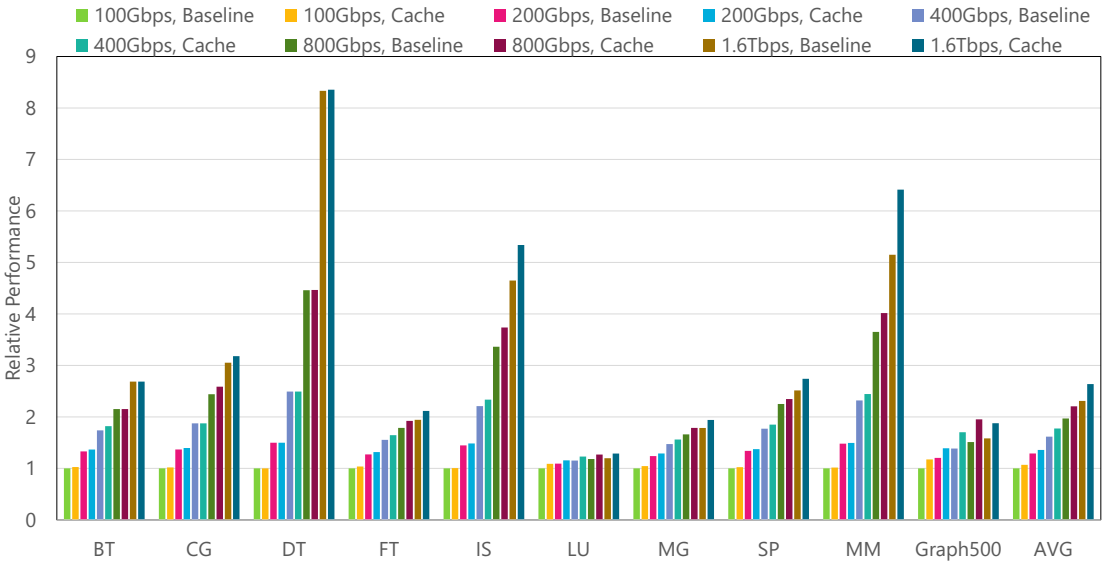


FIGURE 5.28: Relative Application Performance (32 Nodes, Dragonfly).

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 performances, e.g. DT, FT, IS, and MM, with 256 computation nodes 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, because FT, IS, and MM conduct many all-to-all communications and traffic among all network nodes [85].

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 switchable node reduction function becomes effective. high link bandwidth.

Figures 5.27 and 5.28 show the performance improvement of 32 computation node systems with the higher network bandwidth, while the improvement are moderate than that of the 256 computation node systems.

5.5 Hardware Only Approach for Switchable Node Reduction Function

One might think that the cache does not have a strong demand for the switchable node reduction function. It can be implemented by the hardware synthesis optimized to typical network topologies. However, it is costly because a mapping between switch port and network topology can be complex. We support arbitrary patterns of such a mapping. Figure 5.29 is a block diagram of a hardware only approach for switchable node reduction function.

We evaluated the case for hardware only approach for switchable node reduction function. The switchable node reduction function is designed with the Nangate 45nm Open Cell Library [101]. We completed its synthesis using Synopsys Design Compiler O-2018.06-SP4, as well as the evaluation in Chapter 5. Our design only supports up to 16 patterns of the mapping between switch port and network topology. However, its hardware amount is significant as shown in Figure 5.30.

Figure 5.30 illustrated that the hardware amount of the switchable node reduction function without cache is proportional to the number of the mapping patterns. We can conclude that the cache should work together with the switchable node reduction function.

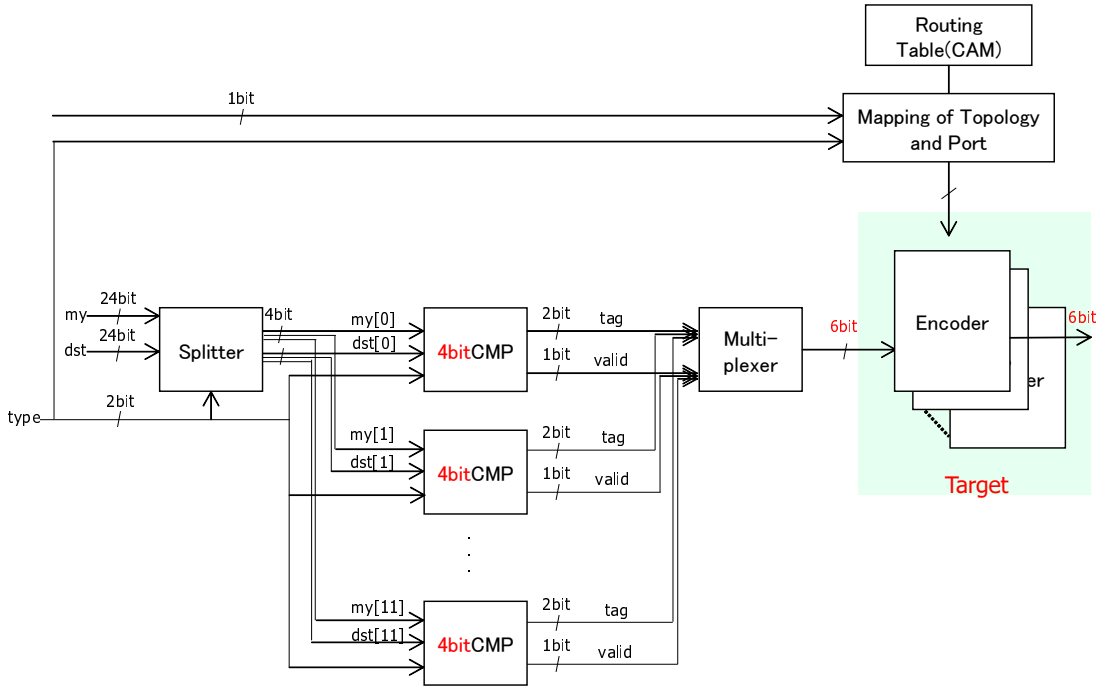


FIGURE 5.29: Block Diagram of Hardware Only Approach for k-ary n-cube Switchable Node Reduction Function.

5.6 Summary

A switch will not able to forward incoming packets at the maximum line rate in interconnection networks. To resolve the latency and throughput problems, an on-chip packet forwarding cache to a switch is explored, as illustrated in the previous Chapter. Since an incoming packet avoids large-latency accessing a TCAM forwarding table if the cache hits, both the line rate and switch delay are much improved. However, the L1 cache size is strictly limited by the chip area. The cache switch would lead to a low hit ratio close to *zero* when the network size is large.

In this Chapter, an additional switchable node reduction function on the packet forwarding cache architecture for elephant-nose large jobs on some specified network topologies is demonstrated. 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. A general node reduction function, which obtains two addresses and their indices and returns a cache tag, is defined to achieve the path regularity. Then, some typical configurations of the switchable node reduction function are stated, which can be derived from the general node reduction function. They required taking custom packet destination addressings on k-ary n-cubes, fat trees, and Dragonfly.

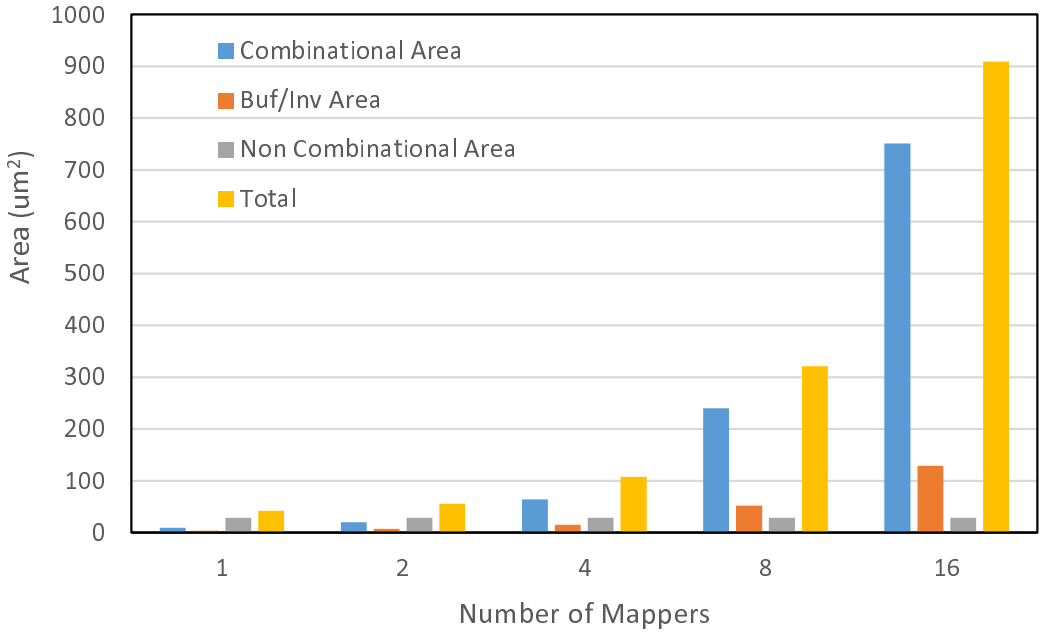


FIGURE 5.30: Hardware Amount of the k-Ary n-Cube Switchable Node Reduction Function without the Cache.

Our evaluation results show that the packet forwarding cache provides up to 933 Gbps line rate We illustrate that parallel applications obtain the gain of 5.07x speed up using the packet forwarding cache.

Chapter 6

Discussions

6.1 Alternative Ways to Achieve 100% Cache Hit Rate

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

6.1.1 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 [104] 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.

6.1.2 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.

6.2 Tradeoff between Hardware Amount and Required Network Size

In Section 5.4, a general mesh/torus hardware reduction function which is equipped with 12 comparators is adopted, as shown in Figure 6.1. It can support 2, 3, and 4 bit width chunks in parallel. The maximum chunk number is $12 = 24\ bit/2\ bit$, when 24 bit address format is used.

With the 2, 3, and 4 bit width chunks, they can address $4(= 2^2)$, $8(= 2^3)$, and $16(= 2^4)$-ary networks, as listed in Table 6.1. With the general hardware reduction function in Figure 6.1, all the three network sizes can be handled, because it has 12 comparators to compare the maximum 12 chunks in parallel.

Table 6.1: Network Size and Chunk Sizes.

Network Size	Chunk Size	Chunk Number	Total Address Length
4-ary, 12-cube	2 bit	12	$24\ bit = 2\ bit \times 12$
8-ary, 8-cube	3 bit	8	$24\ bit = 3\ bit \times 8$
16-ary, 6-cube	4 bit	6	$24\ bit = 4\ bit \times 6$

If up to 16-ary 6-cube network is needed to support in the switch, only six chunks are needed to support in the hardware reduction function. Consequently, the implementation cost can decrease to Figures 6.2, resulting in smaller area overhead and lower latency.

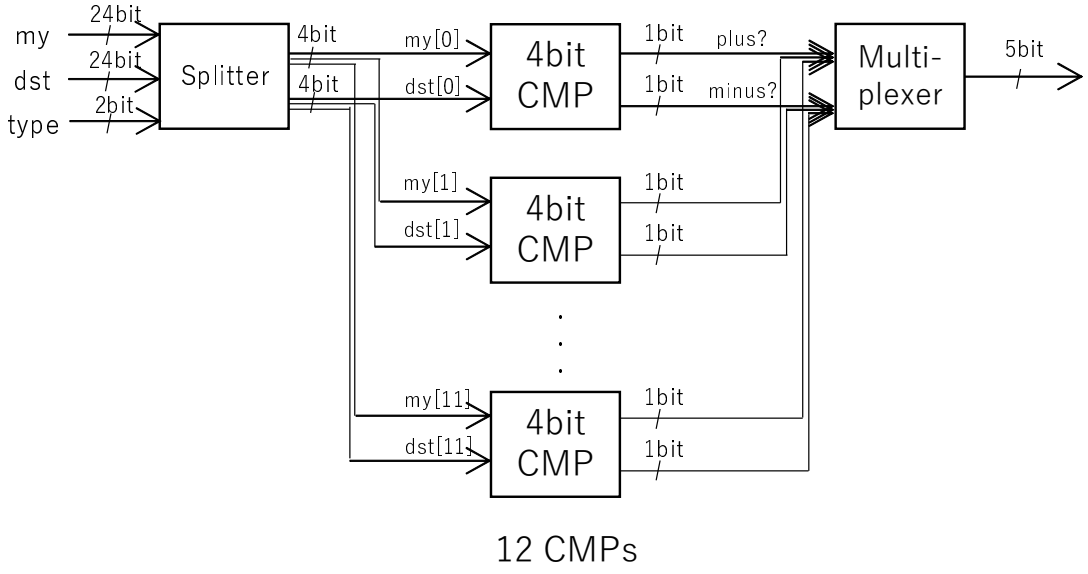


FIGURE 6.1: A General Mesh/Torus Hardware Reduction Function with 12 Comparators.

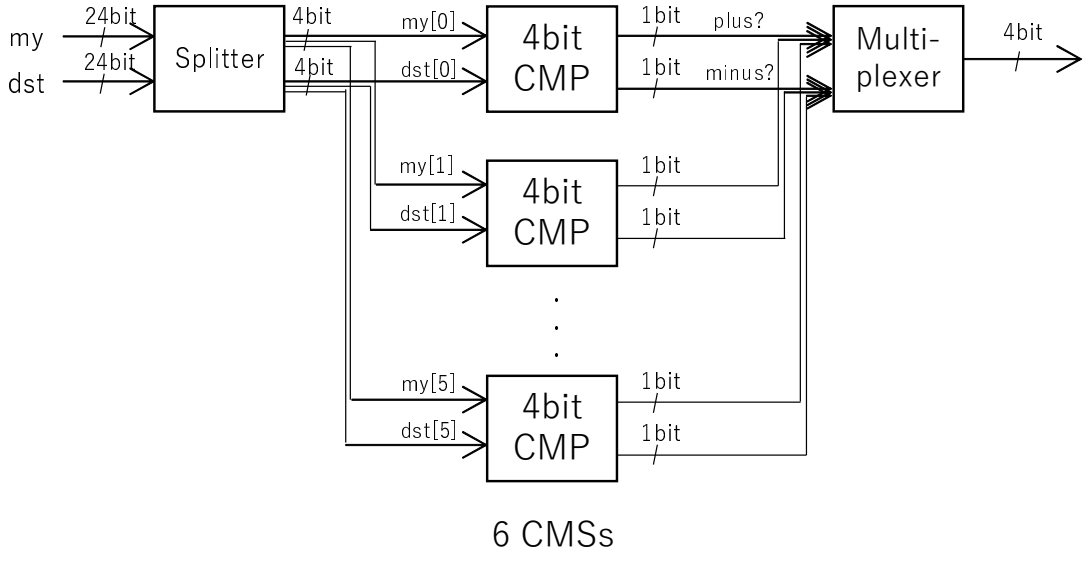


FIGURE 6.2: A Special Mesh/Torus Hardware Reduction Function with Only 6 Comparators.

Chapter 7

Conclusions

7.1 Achievements

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

Our contributions are (1) the four-way set associative packet forwarding cache for up to 2K-node jobs with arbitrary network topologies, and (2) the switchable node reduction function on the cache switch for arbitrary job sizes on typical network topologies.

7.1.1 Four-Way Set Associative Packet Forwarding Cache

The application of packet forwarding cache to all input ports of switches is investigated to improve the switching throughput, the line rate, and the performance of parallel applications. When DWDM(Dense Wavelength Division Multiplexing) is used for the link, the cache can either be installed for each link or each wavelength, depending on the packet throughputs they provides.

The packet forwarding cache area and latency is simulated using the CACTI v6.5 simulator. The transistor model is a high-performance 45 nm process. Since we targeted a 64-port switch, the total cache overhead area becomes $14.6mm^2$ for 2,048 entries

($0.228mm^2$) in a chip. Since switch ASICs have large area, e.g., 329 mm^2 in Tofu of K-computer ICC (65 nm technology), we consider that 2K entries for a 64-port switch are feasible. The CACTI simulation results also showed that the packet forwarding cache supports up to 1.64 Tbps line rate.

In an interconnection network with 512 computation nodes, the conflict miss rate of a 2,048 entry 4-way set associative routing cache is less than 0.1%, with the cache simulation results. The performance of NAS parallel benchmark applications is improved by 4.39x on average in a interconnection network with 256 computation nodes, with MPI simulation results of SimGrid v3.25.

It is concluded that the adoption of the packet forwarding cache increases the communication throughput of interconnection network, the line rate, and the performances of parallel applications. As a result, the packet forwarding cache is strongly recommended to be adopted in HPC commodity switches with arbitrary network topologies.

7.1.2 Switchable Node Reduction Function

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 node reduction function to index a packet forwarding table on a switch. The switchable node reduction 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 switch ASIC, e.g., 51.2 Tbps (800 Gbps $\times$ 64), with the moderate chip-area overhead for incoming shortest packets.

We illustrated 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.

Through this dissertation, it is concluded that a commodity switch should have a packet forwarding cache with switchable node reduction functions. The packet forwarding cache is efficient for forwarding a large number of shortest packets, and the switchable node reduction function is necessary for large scale parallel computers.

7.2 Future Direction and Future Work

7.2.1 Future Interconnection Network

Future interconnection network will rely on commodity technology for Ethernet. Optical technology will be a key factor to follow the 2018 Ethernet roadmap. Ultra high-radix switch ASICs enabled by the optical technology would drastically change the design of interconnection networks.

Co-packaged optics (CPO) [105], i.e., optical technology integration to chip package, is a promising technology for switch ASICs. Hyperscale datacenters highly demand a high-radix high-throughput switch ASIC. BroadCom releases the Ethernet switch ASIC design, Tomahawk 3 [106] (12.8 Tbps) in 2018 and Tomahawk 4 [107] (25.6 Tbps) in 2019. It is expected that a switch ASIC will reach 51.2 Tbps in the 2020s. In current switches, the electric serializer-deserializer (SerDes) conversion consumes significant power, and the broad area of aggregate I/O pluggable ports in creases the onboard wire length.

To mitigate the both problems, the optical technology should be tightly coupled with a switch ASIC. Once the O/E conversion is performed, the data is electrically transferred on a switching ASIC. Thanks to CPO, the line rate continues to increase in the next decade in interconnection network. Thus, the cache switch is more important in the next decade.

Besides the optical technology, the micro switch architecture would change, such as speculation, prediction for decreasing the latency, and look-ahead routing for removing the dependency of the pipeline [24]. The 3-D and 2.5-D chip integration including chiplet also affects the allowable capacity of cache and switch microarchitecture. The cache switch is applicable for the various switch microarchitectures, because it only affects the routing-computation unit on a switch. Our future work is to investigation of the impact of cache switch on them.

7.2.2 Future Software-Defined Network

Software-defined networks(SDN) also continue to increase the line rate. In SDN, packets are forwarded according to the decision of the control plane. Then, the packets are actually forwarded by the data plane. The SDN controller controls all control planes in the network. Consequently, accessing the controller introduces slow packet forwarding rate.

In this context, the cache strategy should care about the rule-dependent analysis for the longest prefix match. At present, TCAM is used as a cache, however, the depth of the cache hierarchy increases as the required line rate increases. That is, L1 cache will be used in the near future, and L1 cache and TCAM will work together. The low-latency consistency model of the longest rule matching is needed. The development of the consistency model is future work to apply a cache switch to future software-defined networks.

Bibliography

[1] Top 500 Supercomputer Sites. http://www.top500.org/.

[2] Ethernet Alliance. https://ethernetalliance.org/.

[3] InfiniBand Trade Association. https://www.infinibandta.org/infiniband-roadmap/.

[4] Anurag Agrawal and Changhoon Kim. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–32. IEEE Computer Society, 2020.

[5] Vladimir Gurevich. Programmable data plane at terabit speeds. *Barefoot Netw., Santa Clara, CA, USA, Tech. Rep*, 2017.

[6] Peter A Boyle. The bluegene/q supercomputer. *PoS LATTICE2012*, 20, 2012.

[7] Yuichiro Ajima, Takahiro Kawashima, Takayuki Okamoto, Naoyuki Shida, Kouichi Hirai, Toshiyuki Shimizu, Shinya Hiramoto, Yoshiro Ikeda, Takahide Yoshikawa, Kenji Uchida, et al. The tofu interconnect d. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 646–654. IEEE, 2018.

[8] Mitsuhisa Sato, Yutaka Ishikawa, Hirofumi Tomita, Yuetsu Kodama, Tetsuya Odajima, Miwako Tsuji, Hisashi Yashiro, Masaki Aoki, Naoyuki Shida, Ikuo Miyoshi, et al. Co-design for a64fx manycore processor and" fugaku". In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[9] Kostas Pagiamtzis and Ali Sheikholeslami. Content-addressable memory (cam) circuits and architectures: A tutorial and survey. *IEEE journal of solid-state circuits*, 41(3):712–727, 2006.

[10] K Scott Hemmert. 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.

[11] NVIDIA. https://www.nvidia.com/en-us/networking/infiniband/qm8700/.

[12] Intel Tofino2. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html, 2021.

[13] Renesas. https://www.renesas.com/us/en/products/memory-logic/network-search-engine/r8a20410bg-g-network-search-engines-nse-s.

[14] RENESAS Electronics Corp. Network Search Engine. https://www.renesas.com/jp/ja/products/memory/application-specific-memory/network-search-e ngine.html.

[15] Yusuke Tanimura, Shinichiro Takizawa, Hirotaka Ogawa, and Takahiro Hamanishi. Building and evaluation of cloud storage and datasets services on ai and hpc converged infrastructure. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1992–2001. IEEE, 2020.

[16] Sadaf R Alam, Ladina Gilly, Colin J McMurtrie, and Thomas C Schulthess. Cscs and the piz daint system. In *Contemporary High Performance Computing*, pages 149–173. CRC Press, 2019.

[17] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[18] Simon McIntosh-Smith, James Price, Andrei Poenaru, and Tom Deakin. Benchmarking the first generation of production quality arm-based supercomputers. *Concurrency and Computation: Practice and Experience*, 32(20):e5569, 2020.

[19] Erik Alerstam, Tomas Svensson, and Stefan Andersson-Engels. Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration. *Journal of biomedical optics*, 13(6):060504, 2008.

[20] Douglas Doerfler, Brian Austin, Brandon Cook, Jack Deslippe, Krishna Kandalla, and Peter Mendygral. Evaluating the networking characteristics of the cray xc-40 intel knights landing-based cori supercomputer at nersc. *Concurrency and Computation: Practice and Experience*, 30(1):e4297, 2018.

[21] Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydın Buluç. High-performance sparse matrix-matrix products on intel knl and multicore architectures. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, pages 1–10, 2018.

[22] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[23] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, 2003.

[24] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[25] HPCI Roadmap. http://www.open-supercomputer.org/workshop/sdhpc/.

[26] Graph Golf: The Order/degree Problem Competition. http://research.nii.ac.jp/graphgolf/.

[27] Eitan Zahavi, Gregory Johnson, Darren J Kerbyson, and Michael Lang. Optimized infinibandtm fat-tree routing for shift all-to-all communication patterns. *Concurrency and Computation: Practice and Experience*, 22(2):217–231, 2010.

[28] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88. IEEE, 2008.

[29] Ioannis Patronas, Angelos Kyriakos, and Dionysios Reisis. Switching functions of a data center top-of-rack (tor). In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 364–367. IEEE, 2016.

[30] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ISCA*, pages 77–88, 2008.

[31] Andrew R. Curtis, Tommy Carpenter, Mustafa Elsheikh, Alejandro López-Ortiz, and S. Keshav. Rewire: An optimization-based framework for unstructured data center network design. In *2012 Proceedings IEEE INFOCOM*, pages 1116–1124, 2012. doi: 10.1109/INFCOM.2012.6195470.

[32] Jayaram Mudigonda, P Yalagandula, and JC Mogul. Taming the flying cable monster: A topology design and optimization fr amework for data-center networks. *USENIX ATC*, pages 1–14, 2011. URL http://static.usenix.org/events/atc11/tech/final_files/Mudigonda.pdf.

[33] Michihiro Koibuchi, Ikki Fujiwara, Hiroki Matsutani, and Henri Casanova. Layout-conscious Random Topologies for HPC Off-chip Interconnect s. In *HPCA*, pages 484–495, 2013.

[34] Ikki Fujiwara, Michihiro Koibuchi, Hiroki Matsutani, and Henri Ca sanova. Skywalk: a Topology for HPC Networks with Low-delay Switches. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 263–272, May 2014.

[35] Steve Scott, Dennis Abts, John Kim, and William J Dally. The blackwidow high-radix clos network. *ACM SIGARCH Computer Architecture News*, 34(2):16–28, 2006.

[36] Harsh Sadawarti, Sony Bansal, and Nirlaip Kaur. A survey on multi-stage interconnection networks. *Int. J. Comput. Sci. Trends Technol*, 3(1):143–151, 2015.

[37] T-Y Feng and S-W Seo. A new routing algorithm for a class of rearrangeable networks. *IEEE Transactions on Computers*, 43(11):1270–1280, 1994.

[38] Ralf Klasing, Burkhard Monien, Regine Peine, and Elena A Stöhr. Broadcasting in butterfly and debruijn networks. *Discrete Applied Mathematics*, 53(1-3):183–197, 1994.

[39] Jose Flich, Tor Skeie, Andres Mejia, Olav Lysne, Pedro Lopez, Antonio Robles, Jose Duato, Michihiro Koibuchi, Tomas Rokicki, and Jose Carlos Sancho. A Survey and Evaluation of Topology Agnostic Deterministic Routing Algorithms. *IEEE Trans. on Parallel Distrib. Syst.*, 23(3):405–425, 2012.

[40] Xin Yuan, Santosh Mahapatra, Wickus Nienaber, Scott Pakin, and Michael Lang. A New Routing Scheme for Jellyfish and Its Performance with HPC Workloads. In *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, pages 36:1–36:11, 2013.

[41] Timothy Mark Pinkston and Sugath Warnakulasuriya. On deadlocks in interconnection networks. In *Proceedings of the 24th annual International symposium on Computer Architecture*, pages 38–49, 1997.

[42] Yoram Ofek and Moti Yung. Efficient mechanism for fairness and deadlock-avoidance in high-speed networks. In *International Workshop on Distributed Algorithms*, pages 192–212. Springer, 1990.

[43] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, and Thomas L. Rodeheffer. Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, October 1991.

[44] Federico Silla and Jose Duato. High-performance routing in networks of workstations with irregular topology. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):699–719, 2000. doi: 10.1109/71.877816.

[45] Mayank Parasar, Hossein Farrokhbakht, Natalie Enright Jerger, Paul V. Gratz, Tushar Krishna, and Joshua San Miguel. Drain: Deadlock removal for arbitrary irregular networks. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 447–460, 2020. doi: 10.1109/HPCA47549.2020.00044.

[46] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[47] Dong Chen, Noel A. Eisley, Philip Heidelberger, Robert M. Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L. Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J. Parker. The ibm blue gene/q interconnection network and message unit. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2011.

[48] Steve Scott, Dennis Abts, John Kim, and William J. Dally. The blackwidow high-radix clos network. In *International Symposium on Computer Architecture (ISCA)*, pages 16–28, 2006.

[49] Yukihiro Nakagawa, Takeshi Shimizu, Yoichi Koyanagi, Osamu Shiraki, Shinji Kobayashi, Kazuki Hyoudou, Takashi Miyoshi, Yuuki Ogata, Yasushi Umezawa, Takeshi Horie, et al. A single-chip, 10-gigabit ethernet switch lsi for energy-efficient blade servers. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 404–411. IEEE, 2010.

[50] S. Nishimura, T. Kudoh, H. Nishi, J. Yamamoto, R. Ueno, K. Harasawa, S. Fukuda, Y. Shikichi, S. Akutsu, K. Tasho, and H. Amano. Rhinet-3/sw: an 80-gbit/s high-speed network switch for distributed parallel computing. In *HOT 9 Interconnects. Symposium on High Performance Interconnects*, pages 119–123, 2001. doi: 10.1109/HIS.2001.946703.

[51] Pankaj Gupta, Steven Lin, and Nick McKeown. Routing lookups in hardware at memory access speeds. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, volume 3, pages 1240–1247. IEEE, 1998.

[52] Tzi-cker Chiueh and Prashant Pradhan. High-performance ip routing table lookup using cpu caching. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and*

Communications Societies. The Future is Now (Cat. No. 99CH36320), volume 3, pages 1421–1428. IEEE, 1999.

[53] Jorge García-Vidal, Maribel March, Llorenç Cerdà, Jesús Corbal, and Mateo Valero. A dram/sram memory scheme for fast packet buffers. *IEEE Transactions on Computers*, 55(5):588–602, 2006.

[54] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Wolfgang E. Nagel. Cache coherence protocol and memory performance of the intel haswell-ep architecture. In *2015 44th International Conference on Parallel Processing*, pages 739–748, 2015. doi: 10.1109/ICPP.2015.83.

[55] Michael Attig, Sarang Dharmapurikar, and John Lockwood. Implementation results of bloom filters for string matching. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 322–323. IEEE, 2004.

[56] Giorgos Papadopoulos and Dionisios Pnevmatikatos. Hashing+ memory= low cost, exact pattern matching. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 39–44. IEEE, 2005.

[57] Ioannis Sourdis, Dionisios Pnevmatikatos, Stephan Wong, and Stamatis Vassiliadis. A reconfigurable perfect-hashing scheme for packet inspection. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 644–647. IEEE, 2005.

[58] Kun Huang, Linxuan Ding, Gaogang Xie, Dafang Zhang, Alex X Liu, and Kave Salamatian. Scalable tcam-based regular expression matching with compressed finite automata. In *Architectures for Networking and Communications Systems*, pages 83–93. IEEE, 2013.

[59] Changhoon Kim, Matthew Caesar, Alexandre Gerber, and Jennifer Rexford. Revisiting route caching: The world should be flat. In *Passive and Active Network Measurement, 10th International Conference, PAM*, volume 5448 of *Lecture Notes in Computer Science*, pages 3–12, 2009.

[60] Nadi Sarrar, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang. Leveraging zipf's law for traffic offloading. *Comput. Commun. Rev.*, 42(1):16–22, 2012.

[61] 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)*, number 110, pages 1–6, 2018.

[62] Tariq Javid, Tehseen Riaz, and Asad Rasheed. A layer2 firewall for software defined network. In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pages 39–42. IEEE, 2014.

[63] Naga Praveen Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Cacheflow: Dependency-aware rule-caching for software-defined networks. In Brighten Godfrey and Martín Casado, editors, *Proceedings of the Symposium on SDN Research, SOSR*, page 6, 2016.

[64] Layong Luo, Gaogang Xie, Steve Uhlig, Laurent Mathy, Kavé Salamatian, and Yingke Xie. Towards tcam-based scalable virtual routers. In Chadi Barakat, Renata Teixeira, K. K. Ramakrishnan, and Patrick Thiran, editors, *Conference on emerging Networking Experiments and Technologies, CoNEXT'12*, pages 73–84, 2012.

[65] Michitaka Okuno, Shinji Nishimura, Shinichi Ishida, and Hiroaki Nishi. Cache-based network processor architecture: Evaluation with real network traffic. *IEICE Transactions on Electronics*, E89-C(11):1620–1628, Nov 2006.

[66] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X Liu, Qi Li, and Laurent Mathy. Guarantee ip lookup performance with fib explosion. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 39–50, 2014.

[67] Kyosuke Tanaka, Hayato Yamaki, Shinobu Miwa, and Hiroki Honda. Multi-level packet processing caches. In *2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pages 1–3, 2019. doi: 10.1109/CoolChips.2019.8721336.

[68] Michihiro Koibuchi, Shinichi Ishida, and Hiroaki Nishi. The impact of routing cache on high-performance switches. In *Proc. of the 10th International Conference on Optical Internet (COIN)*, page Tuj2(2 pages), May 2002.

[69] Shinichi Ishida, Michihiro Koibuchi, and Hiroaki Nishi. A case for routing cache on hpc switches. *IEICE Communications Express*, 1(1):49–53, 2012.

[70] Veronica Melesse Vergara, Reuben Budiardja, Paul Peltz, Jeffery Niles Jr, Christopher Zimmer, Dan Dietz, Christopher Fuson, Hong Liu, Paul Newman III, James Simmons, et al. A step towards the final frontier: Lessons learned from acceptance testing of the first hpe/cray ex 3000 system at ornl. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2021.

[71] Maho Nakata. All about ricc: Riken integrated cluster of clusters. In *2011 Second International Conference on Networking and Computing*, pages 27–29, 2011. doi: 10.1109/ICNC.2011.14.

[72] Jean-Michel Morey. Numerical simulation at cea. *Proceedings of SNA+ MC*, 2013.

[73] Miroslav Ruda, Zdeněk Šustr, Jiří Sitera, David Antoš, Lukáš Hejtmánek, Petr Holub, et al. Virtual clusters as a new service of metacentrum, the czech ngi. 2010.

[74] Dror G Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74 (10):2967–2982, 2014.

[75] Klusáček, Dalibor and Chlumskỳ, Václav. Evaluating the Impact of Soft Walltimes on Job Scheduling Performance. In *Job Scheduling Strategies for Parallel Processing*, pages 15–38, May 2018.

[76] George Almasi. Pgas (partitioned global address space) languages., 2011.

[77] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. Introducing openshmem: Shmem for the pgas community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, pages 1–3, 2010.

[78] Torsten Hoefler and Timo Schneider. Optimization principles for collective neighborhood communications. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE, 2012.

[79] Jens Carsten Jackwerth. Generalized binomial trees. *Journal of Derivatives*, 5(2): 7–17, 1996.

[80] Huan Zhou, Vladimir Marjanovic, Christoph Niethammer, and José Gracia. A bandwidth-saving optimization for mpi broadcast collective operation. In *2015 44th International Conference on Parallel Processing Workshops*, pages 111–118, 2015. doi: 10.1109/ICPPW.2015.20.

[81] Rajeev Thakur and William D Gropp. Improving the performance of collective operations in mpich. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 257–267. Springer, 2003.

[82] Ahmad Faraj and Xin Yuan. Automatic generation and tuning of mpi collective communication routines. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 393–402, 2005.

[83] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. Characterization of mpi usage on a production supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 386–400, 2018. doi: 10.1109/SC.2018.00033.

[84] The NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[85] Fabien Chaix, Ikki Fujiwara, and Michihiro Koibuchi. Suitability of the random topology for hpc applications. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 301–304, 2016. doi: 10.1109/PDP.2016.10.

[86] John Shalf, Shoaib Kamil, Leonid Oliker, and David Skinner. Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect. In *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 1–13, 2005. doi: 10.1109/SC.2005.12.

[87] Daniele De Sensi, Salvatore Di Girolamo, Kim H McMahon, Duncan Roweth, and Torsten Hoefler. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[88] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP laboratories*, 27:28, 2009.

[89] Takashi Toyoshima. Icc: An interconnect controller. *Tofu interconnect architecture, HOT CHIPS*, 22, 2010.

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

[91] Samoda Gamage and Ajith Pasqual. High performance parallel packet classification architecture with popular rule caching. In *2012 18th IEEE International Conference on Networks (ICON)*, pages 52–57. IEEE, 2012.

[92] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. A Case for Random Shortcut Topologies for HPC Interconnects. In *ISCA*, pages 177–188, 2012.

[93] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. In *NSDI*, pages 225–238, 2012.

[94] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. Small-World Datacenters. In *ACM Symposium on Cloud Computing*, page 2, 2011.

[95] M. Ould-Khaoua. A performance model for duato's fully adaptive routing algorithm in k-ary n-cubes. *IEEE Transactions on Computers*, 48(12):1297–1304, 1999.

[96] Banit Agrawal and Timothy Sherwood. Ternary cam power and delay model: Extensions and uses. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(5):554–564, 2008. doi: 10.1109/TVLSI.2008.917538.

[97] S. Matsunaga, N. Sakimura, R. Nebashi, Y. Tsuji, A. Morioka, T. Sugibayashi, S. Miura, H. Honjo, K. Kinoshita, H. Sato, S. Fukami, M. Natsui, A. Mochizuki, S. Ikeda, T. Endoh, H. Ohno, and T. Hanyu. Fabrication of a 99%-energy-less nonvolatile multi-functional cam chip using hierarchical power gating for a massively-parallel full-text-search engine. *Symposium on VLSI Circuits*, pages C106–C107, 2013.

[98] Tolga Soyata and John Liobe. pbcam: Probabilistically-banked content addressable memory. In *2012 IEEE International SOC Conference*, pages 27–32, 2012. doi: 10.1109/SOCC.2012.6398372.

[99] Telajala Venkata Mahendra, Sandeep Mishra, and Anup Dandapat. Self-controlled high-performance precharge-free content-addressable memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(8):2388–2392, 2017. doi: 10.1109/TVLSI.2017.2685427.

[100] Sandeep Mishra, Telajala Venkata Mahendra, and Anup Dandapat. A 9-t 833-mhz 1.72-fj/bit/search quasi-static ternary fully associative cache tag with selective matchline evaluation for wire speed applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(11):1910–1920, 2016. doi: 10.1109/TCSI.2016.2592182.

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

[102] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

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

[104] MPI Forum. https://www.mpi-forum.org/mpi-30/.

[105] P Maniotis, Laurent Schares, Benjamin G Lee, Marc A Taubenblatt, and Daniel M Kuchta. Scaling hpc networks with co-packaged optics. In *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2020.

[106] Wei Bai, Kai Chen, Shuihai Hu, Kun Tan, and Yongqiang Xiong. Congestion control for high-speed extremely shallow-buffered datacenter networks. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 29–35, 2017.

[107] Nikos Terzenidis, Apostolos Tsakyridis, G Giamougiannis, Miltiadis Moralis-Pegios, Konstantinos Vyrsokinos, and Nikos Pleros. A 25.6 tbps capacity 1024-port

hipoλaos optical packet switch architecture for disaggregated datacenters. In *Optical Fiber Communication Conference*, pages W1F–4. Optical Society of America, 2020.