Received February 26, 2020, accepted March 15, 2020, date of publication March 17, 2020, date of current version April 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2981540

Packet Processing Architecture Using Last-Level-Cache Slices and Interleaved 3D-Stacked DRAM

TOMOHIRO KORIKAWA[1], AKIO KAWABATA[1], FUJUN HE[2], (Student Member, IEEE), AND EIJI OKI[2], (Fellow, IEEE)
[1]Network Service Systems Laboratories, NTT Corporation, Tokyo 180-8585, Japan
[2]Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

Corresponding author: Tomohiro Korikawa (tomohiro.koorikawa.xa@hco.ntt.co.jp)

ABSTRACT Packet processing performance of Network Function Virtualization (NFV)-aware environment depends on the memory access performance of commercial-off-the-shelf (COTS) hardware systems. Table lookup is a typical example of packet processing, which has a significant dependence on memory access performance. Thus, the on-chip cache memories of the CPU are becoming more and more critical for many high-performance software routers or switches. Moreover, in the carrier network, multiple applications run on top of the same hardware system in parallel, which requires the capacity of cache memories. In this paper, we propose a packet processing architecture that enhances memory access parallelism by combining on-chip last-level-cache (LLC) slices and off-chip interleaved 3 Dimensional (3D)-stacked Dynamic Random Access Memory (DRAM) devices. Table entries are stored in the off-chip 3D-stacked DRAM, so that memory requests are processed in parallel by using bank interleaving and channel parallelism. Also, cached entries are distributed to on-chip LLC slices according to a memory address-based hash function so that each CPU core can access on-chip LLC in parallel. The evaluation results show that the proposed architecture reduces the memory access latency by 62 % and 12 % and increases the throughput by 108 % and 2 % with reducing blocking probability of memory requests 96 % and 50 %, compared to the architecture with on-chip shared LLC and that without on-chip LLC, respectively.

INDEX TERMS Communication systems, memory architecture, network function virtualization, performance analysis, queueing analysis.

I. INTRODUCTION

Packet processing performance of Network Function Virtualization (NFV)-aware environment depends on the memory access performance of commercial-off-the-shelf (COTS) hardware systems. Table lookup is a typical example of packet processing, which has a significant dependence on memory access performance.

Today's COTS hardware and virtualization technology reduce the number of memory accesses associated with the data transfers between the network interface card (NIC) and the applications. Intel Data Plane Development Kit (DPDK) [1] and Single Root I/O Virtualization (SR-IOV) [2] are well-known examples of recent virtualization technologies for fast packet processing.

The on-chip cache memories of the CPU are becoming more critical for high-performance software routers or switches. While the recent virtualization technologies reduce the number of memory accesses, lowering memory access latency is also required in order to increase packet processing performance. Several software-based packet processing applications that intensively use on-chip cache memories to achieve more than tens of Gbps [4], [5].

In the carrier network, multiple virtualized network functions (VNFs) are deployed in the same COTS hardware system, which requires the capacity of cache memories. There are various network functions in the carrier network. For the broadband connection service, the Broadband Network Gateway (BNG) function is needed to terminate Point to Point

The associate editor coordinating the review of this manuscript and approving it for publication was Yiming Huo.

Protocol (ppp) sessions from end-users. Firewall or DDoS mitigation functions are necessary to protect users from cyber attacks. The Evolved Packet Core (EPC) functions accommodate mobile data traffic. These network functions consist of several packet processing elements such as parsing, classification, editing, and metering, each of which requires table lookup and memory accesses. When these network functions are running on the same hardware system, usually called the multi-tenant environment, multiple applications issues many memory accesses from each corresponding CPU core in parallel. This situation requires both speed and capacity of cache memories for high-performance packet processing.

In current COTS hardware architecture, the carrier-scale packet processing applications cannot allocate sufficient capacity of on-chip LLC. Moreover, the memory capacity of the on-chip cache is dependent on the physical space of the semiconductor chip. The insufficient capacity of on-chip cache memories and bandwidth of off-chip memory have been major problems of modern computer architecture [51]–[56]. Thus, off-chip fast memory that has larger memory capacity is required so that large data such as lookup tables for virtualized carrier network functions can be accommodated. The work in [24] presented a packet processing architecture that uses 3 Dimensional (3D)-stacked Dynamic Random Access Memory (DRAM) device as an off-chip last-level-cache (LLC) instead of an on-chip shared LLC. The 3D-stacked DRAM with more memory parallelism as an off-chip LLC outperforms the on-chip shared LLC with less memory parallelism. This result shows that 3D-stacked DRAM provides both faster memory accesses and larger memory capacity than on-chip shared LLC in terms of the number of memory requests that can be processed in a certain period of time.

The on-chip shared LLC of modern multi-core CPU usually comprises multiple slices, each of which belongs to a CPU core. Each LLC slice can be accessed in parallel, which increases the memory access parallelism. In the multi-tenant environment, the operator of a system assigns computing resources such as CPU cores, memory capacities, and LLC slices to a particular application. The number of assigned LLC slices is configured by using functions such as Intel Cache Allocation Technology (CAT) [28]. When some of the LLC slices are assigned to an application, the rest of LLC slices can be allocated to other applications running in the same hardware. A packet processing architecture that uses on-chip LLC slices and off-chip 3D-stacked DRAM may increase the memory parallelism. The relation between the packet processing performance and the number of assigned LLC slices needs to be understood when a multi-core CPU with LLC slices is combined with the off-chip 3D-stacked DRAM before we build the real system. We consider that a better understanding of the dependency on the number of assigned resources, including LLC slices, leads to more efficient resource allocation in a multi-tenant NFV environment. There is no work that evaluates the performance dependency of the proposed architecture on the number of assigned resources when combining the LLC slices with 3D-stacked DRAM.

This paper proposes a packet processing architecture that enhances memory access parallelism by combining on-chip LLC slices and off-chip 3D-stacked DRAM devices. Table entries are stored in the off-chip 3D-stacked DRAM, so that memory requests are processed in parallel by using bank interleaving and channel parallelism. Also, cached entries are distributed to on-chip LLC slices according to a memory address-based hash function so that each CPU core can access on-chip LLC in parallel. The evaluation results show that the proposed architecture reduces the memory access latency by 62 % and 12 % and increases the throughput by 108 % and 2 % with reducing blocking probability of memory requests 96 % and 50 %, compared to the architecture with shared on-chip LLC and that without on-chip LLC, respectively. These results indicate that while the memory access parallelism of on-chip LLC slices improves the packet processing performance, the memory access parallelism of off-chip 3D-stacked DRAM can improve the packet processing performance. The proposed architecture increases the performance of carrier-scale packet processing applications in a multi-tenant NFV environment.

The rest of this paper is organized as follows. Section II describes the basic mechanism of the memory system and the characteristics of memory devices. Section III presents the proposed architecture. Section IV describes the system modeling of the proposed architecture. Section V presents the performance evaluation. Section VI describes related work. Section VII discusses on the limitations and the future directions. Finally, Section VIII concludes this paper.

II. BACKGROUND

A. MEMORY SYSTEM IN COTS SYSTEM

Figure 1 shows the DRAM system in today's COTS hardware system. The DRAM system is composed of memory requestors, DRAM controllers, and DRAM memory devices. Usually, a memory requestor is a CPU or a Direct Memory Access (DMA). The memory requestor issues memory commands, such as *read* or *write*. According to the memory commands, the DRAM controller physically accesses DRAM devices.

There are two types of buses that connect a memory controller and memory devices. The DRAM devices are connected to the DRAM controllers via command buses and data buses. Memory commands are transferred through a command bus. A data bus is used for data transfer between a memory controller and memory devices. A command bus and a data bus are used independently. Thus, a memory requestor can use the data bus while the memory requestor issues memory commands at the same time.

DRAM devices and a DRAM controller are connected via multiple memory channels. Each memory channels is used independently, which means that a DRAM controller can access DRAM devices in parallel via multiple memory channels. A memory channel comprises multiple banks. Bank

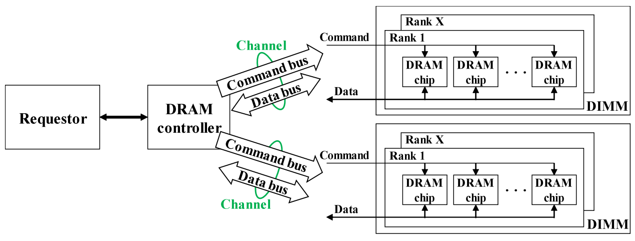


FIGURE 1. DRAM sytem overview.

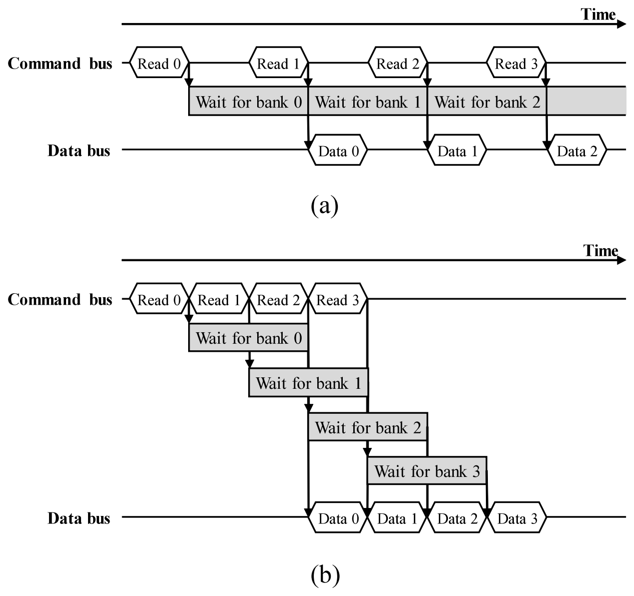


FIGURE 2. Diagrams of command bus and data bus in DRAM system. (a) Without bank interleaving. (b) With bank interleaving.

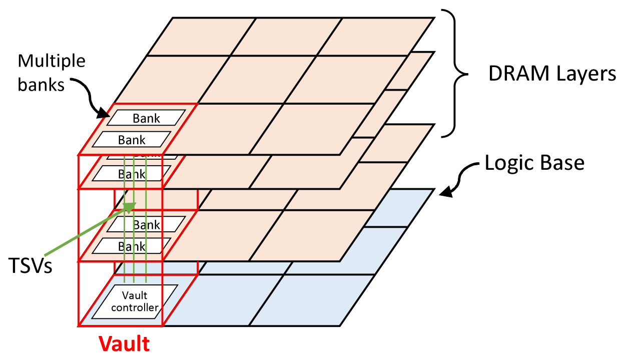


FIGURE 3. Schematic structure of hybrid memory cube.

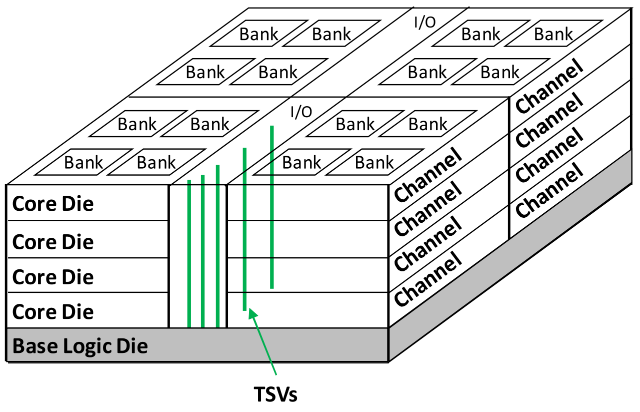


FIGURE 4. Schematic structure of high bandwidth memory.

interleaving is a well-known technique, where multiple banks in a channel can be accessed at the same time unless there is a collision of commands or data in each bus. As shown in Figure 2, more data can be read from DRAM banks in less time by using bank interleaving.

B. 3D-STACKED DRAM

3D-stacked DRAM is an emerging type of memory device that consolidates multiple DRAMs in a single memory device. It consists of vertically stacked DRAM layers, each of which is connected by using Through Silicon Via (TSV) technology so that memory requestors can access every DRAM layer. Conventional Double Data Rate × (DDRx) DRAM devices require more area inefficient wires between a processor and memory devices than 3D-stacked DRAMs, which prevents a CPU from using more memory channels. The higher density of memory channels in the 3D-stacked DRAMs enables CPU to be connected with more number of memory channels, compared to using conventional DDRx DRAM devices. Thus 3D-stacked DRAM provides more memory channels without losing versatility of conventional DDRx DRAM devices. This is why we use an off-chip 3D-stacked DRAM in the proposed architecture.

Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) are well-known and commercially available examples of 3D-stacked DRAM devices. Both HMC and HBM consolidate conventional DRAMs in a single memory device, which enhances memory access parallelism compared to the conventional DRAM devices by consolidating more memory channels and banks.

Figure 3 shows the structure of HMC. HMC comprises several DRAM layers on top of the bottom layer, the logic base [41]. A *vault* is a vertical unit, which corresponds to what is called a channel in traditional DRAM devices. Each vault is accessible in parallel. Each DRAM layer of a vault has several banks. In the logic base, simple arithmetic operations can be performed to the data in the DRAM layers, which is utilized in Processing-in-Memory (PIM) architecture [35], [36]. The memory controller of HMC and the logic base are connected through several high-speed serial links.

Figure 4 shows the schematic structure of HBM. As with HMC, HBM has several Core DRAM Dies (DRAM layers), each of which comprises conventional DRAM that is connected through TSV. HBM has a wider I/O bus compared to HMC to enhance the memory bandwidth. The bottom layer of HBM, called base logic die [25].

Both HMC and HBM have several channels and banks, which allows us to model them as parallel DRAM devices.

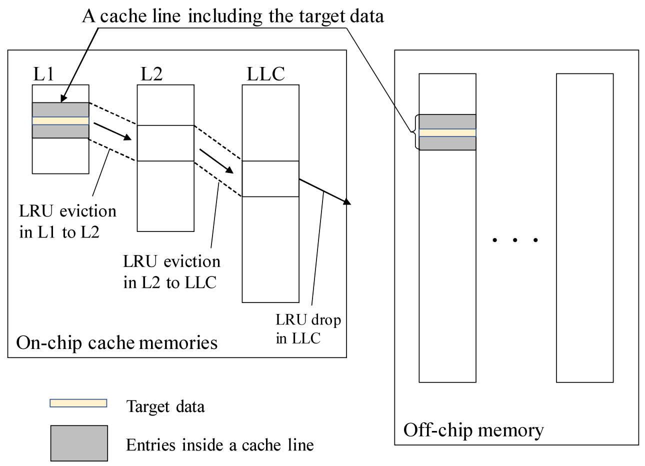


FIGURE 5. **Mechanism of cache memory systems.**

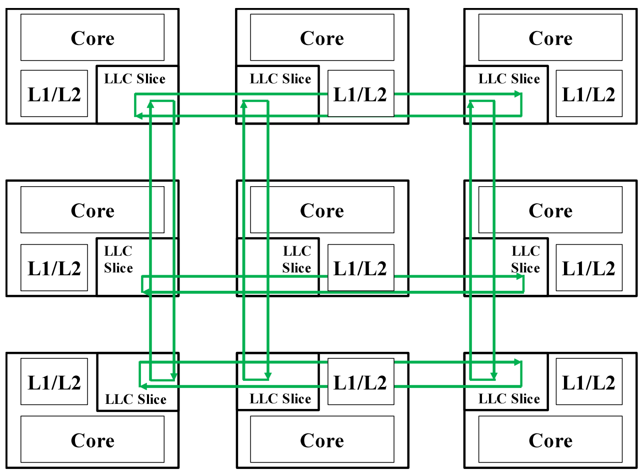


FIGURE 6. **Architecture of LLC that consist of LLC slice connected via mesh inter-core bus among CPU cores. The inter-core bus is shown in green lines.**

C. CACHE MEMORY SYSTEM

There are several levels of on-chip cache memories in a multi-core CPU. Each CPU core has one or two levels of its dedicated cache memories, usually called level 1 (L1) cache and level 2 (L2) cache. A last-level-cache (LLC) is shared among every CPU core inside the same CPU. Usually, the L1 cache is the fastest and has the smallest memory capacity; on the other hand, LLC is the slowest with the largest memory capacity.

Figure 5 shows the mechanism of a cache memory system including off-chip memory devices. A *cache line* is an elementary block of data transferred between the cache and the off-chip memory. Usually, the data physically around a particular data, shown as the target data in Figure 5, is likely to be accessed next, which is known as data spatial locality of the data. By assuming the space locality of data, cache lines improve the hit probabilities of cache memories.

If a cache line including the target data that corresponds to a request is found in a certain level of cache, which is called a *hit*, the cache returns the corresponding cache line of the request to the CPU core that issued the request. If any cache line including the target data is not found in a certain level of cache, which is called a *miss*, the request accesses the next level of cache or the off-chip memory until the request finds the corresponding cache line. A cache line is replaced so that a newly loaded cache line can be accommodated in the cache. The policy that decides which cache line is replaced next is defined in the system. Typically, the least-recently-used (LRU) policy is used, where the LRU cache line in the L1 cache or L2 cache is evicted to the L2 cache or LLC, and the LRU cache line in LLC is dropped.

D. ARCHITECTURE OF LAST-LEVEL-CACHE

Recent Intel's microarchitecture later than SandyBridge has LLC that comprises distributed slices, each of which belongs to each CPU core, as shown in Figure 6. Every CPU core including its LLC slice is connected via an inter-core bus that is usually a bi-directional ring or mesh architecture [27], [31]. According to [65], Intel's Skylake generation processor actually introduces mesh-interconnect among on-chip components. The multiple LLC slices and the interconnect among them may make the memory system more complicated than those of the shared LLC that consists of a single LLC slice. A power and area efficient router architecture for a 2D mesh interconnect among CPU cores and LLC slices of each CPU core was presented in [62]. Thus, although the detail of the microarchitecture of the CPUs is not publicly known, we consider that today's multi-core CPU utilizes such energy and area efficient technology to implement the interconnects among CPU cores and their LLC slices. Additionally, the analysis of power and area of an on-chip LLC was presented in [67].

Although the physical LLC slices are separated, each LLC slice is addressable, which enables each CPU core to access every LLC slice as a single logical LLC. Each LLC slice has the same capacity as the other LLC slices in the CPU. Thus the capacity of the logical LLC equals the product of the number of CPU cores and the capacity of an LLC slice. Each LLC slice can be accessed in parallel from different CPU cores unless multiple CPU cores access the same LLC slice simultaneously, which improves the effective memory bandwidth of LLC.

Memory addresses of data and requests are mapped to each LLC slices according to a hash-function-based rule so that the number of requests to each LLC slice can be evenly balanced. The detail of the hash function is usually undocumented, while the mapping is known to be conducted by a calculation based on a particular part of the physical memory address of a data or a request. Thus several studies reverse-engineered the hash function of recent CPUs [29], [30]. Figure 7 shows an example of memory address mapping to LLC slices. In this example, the memory address range of each cache line is mapped to one of the LLC slices evenly. The memory address range of n_{CL}-th cache line is mapped to $(n_{\mathrm{CL}} \mod N_{\mathrm{Slice}} + 1)$-th LLC slice, where N_{Slice} represents the total number of LLC slices in this example. The hash function of memory

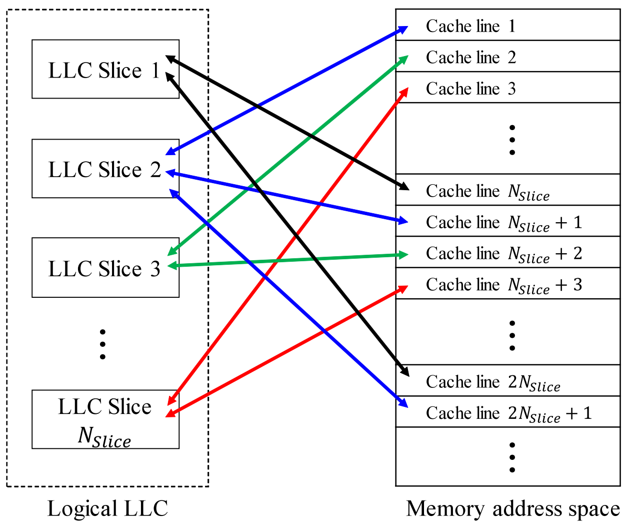


FIGURE 7. Example of address mapping to LLC slices. The memory address range of n_{CL}-th cache line is mapped to (n_{CL} mod $N_{\mathrm{Slice}}+1$)-th LLC slice.

address mapping decides which address range of cache line is to be stored in which LLC slice and to which LLC slice a request that misses L2 cache is transferred.

In NFV-aware, multi-tenant environment, various applications run in the same hardware system simultaneously. Sometimes, applications that require frequent memory accesses, usually called memory-intensive applications, dominantly use LLC slices including LLC slices that belong to the other CPU cores. These applications are usually called noisy neighbors as they consume extra LLC slices in which the other applications are to allocate. Regarding this problem, LLC slices are also becoming one of the computing resources as well as the other resources such as the CPU cores and memory capacity. For the assignment of the LLC slices to each CPU core that runs a certain process, several slice-aware memory management technologies such as Intel Cache Allocation Technology (CAT) [28] are becoming popular in the operation of NFV infrastructure [29].

III. PROPOSED ARCHITECTURE

Figure 8 shows the proposed architecture. It consists of a multi-core CPU, a Field Programmable Gate Array (FPGA), a 3D-stacked DRAM, a DRAM, and network interfaces. Each CPU core has its dedicated L1 and L2 cache memories. The LLC of the multi-core CPU comprises multiple LLC slices. The performance counter counts the statistics related to the performance of the CPU such as the number of LLC misses. An FPGA connects a CPU and an off-chip 3D-stacked DRAM. An external DRAM is used as a packet buffer.

Each incoming packet is processed as follows. (1) A packet that comes from the network interface is directly transferred to the DRAM by using DMA, where the packet is buffered in the packet buffer. The packet is randomly assigned to one of the CPU cores. (2) The assigned CPU core reads the header information of the packet from the packet buffer in the DRAM. Then, the CPU core issues memory requests to read table entries stored in the cache memories and the 3D-stacked DRAM in order to decide the next action for the packet. (3) The CPU core sends the packet including its payload from the packet buffer in the DRAM through the network interface.

In the proposed architecture, both on-chip LLC slices and off-chip 3D-stacked DRAM enhance the memory access parallelism to increase the packet processing performance. The distribution of table data in the 3D-stacked DRAM was presented in [6], [26]. Table data is split into several partial tables as many as the number of banks in a channel. Each partial table is allocated to each bank so that the partial tables comprise the original table in the channel. Then, the copy of the data in the channel is placed in the rest of the channels in the 3D-stacked DRAM. The type of data stored in the 3D-stacked DRAM is not limited to lookup tables. Information on other packet processing such as Quality of Service (QoS) controlling function can be stored in the 3D-stacked DRAM.

An FPGA accommodates the circuits of the following functions: a memory controller of the 3D-stacked DRAM, the skip selection logic, and a hash-function-based distributor of memory requests. The distributor distributes memory requests to the appropriate channel/bank sets so that the number of interleaved banks in each channel is minimum. The distributor has an internal table in the FPGA that records the state of each channel/bank set, which is utilized to determine the appropriate channel/bank set. As presented in [6], [26], the state of channel/bank is idle or busy with w-interleaving, where w is the number of interleaved bank in a channel.

Several semiconductor companies have already released the IP core products of the 3D-stacked DRAM controller for FPGAs [57]–[61]. Additionally, Intel released an FPGA that has integrated HBMs in a single device package, which can utilize the maximum bandwidth of up to two HBM devices [71]. Also, this FPGA-based product may make it easier to use the 3D-stacked DRAMs with less hardware modification from today's COTS hardware, compared to newly designing and implementing the hard-coded logic. The processing latency of FPGA, which is based on Static Random Access Memory (SRAM), is sufficiently smaller than the DRAM access latency. Therefore the memory parallelism and bandwidth are more critical metric than the additional latency introduced by the FPGA. Thus we assume that an FPGA is a good candidate to use the 3D-stacked DRAM. Recent CPUs have inter-chip links such as Intel Quick Path Interconnect (QPI) or Ultra Path Interconnect (UPI). These inter-chip links can be used to connect the CPU and the FPGA [7]–[9].

An operator of the proposed architecture assigns a certain number of CPU cores and LLC slices to an application. If none of the LLC slices is assigned to an application, the application skips accessing the on-chip LLC. In other words, the application directly accesses off-chip 3D-stacked DRAM when there is a miss in the L2 cache. Also, the operator sets the threshold of the LLC miss rate to the skip selection logic in the FPGA. The skip selection logic determines whether an application should skip the on-chip LLC slices

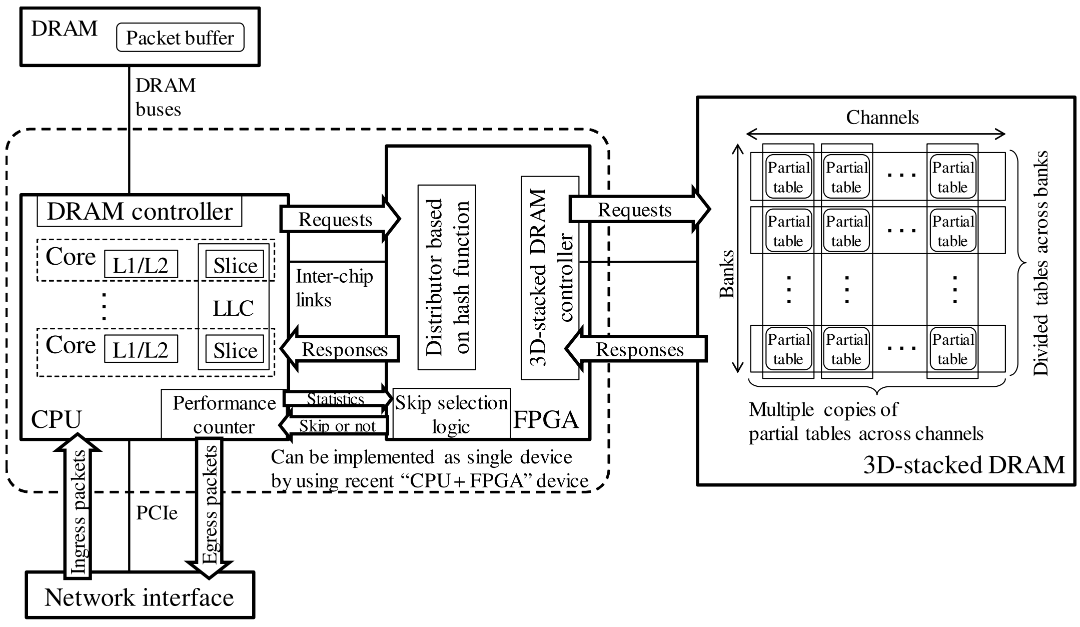
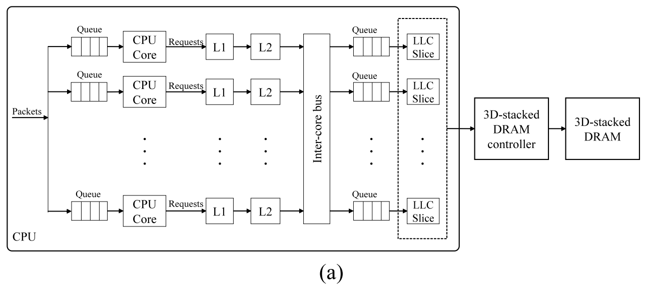


FIGURE 8. Proposed architecture.



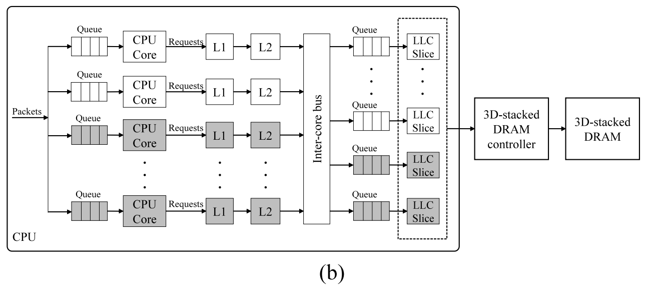


FIGURE 9. System model of proposed architecture with on-chip logical LLC which physically consists of multiple LLC slices. (a) System model of proposed architecture where all the CPU cores and LLC slices are assigned. (b) System model of architecture where the numbers of assigned CPU cores and LLC slices are less than the numbers of CPU cores and LLC slices that the system has. The gray queues, CPU cores, and LLC slices are not assigned.

by comparing the statistics of the LLC miss rate measured in the performance counter of the CPU with the threshold. We explain this feature in Section V-D.

IV. SYSTEM MODEL

A. SYSTEM MODEL OF PROPOSED ARCHITECTURE

Figure 9 shows the system models of the proposed architecture for two example cases. Both models have multiple CPU cores in the CPU, 3D-stacked DRAM that is connected to the CPU via a 3D-stacked DRAM controller. Each CPU core has a queue and dedicated level 1 (L1) and level 2 (L2) caches. The on-chip LLC comprises LLC slices, each of which has a queue. The queues of LLC slices and all the CPU cores are connected via inter-core bus so that each CPU core can access every LLC slice.

Let the system has C_{sys} CPU cores and L_{sys} LLC slices. The operator of the system assigns some of the C_{sys} CPU cores and L_{sys} LLC slices to a specific application. We define the number of assigned CPU cores and LLC slices to the application as C and L, where $0 \leq C \leq C_{sys}$ and $0 \leq L \leq L_{sys}$, respectively. Figure 9(a) shows the system model of proposed architecture where $C = C_{sys}$ CPU cores and $L = L_{sys}$ LLC slices in the system are assigned. Figure 9(b) shows system model of proposed architecture where $C < C_{sys}$ CPU cores and $L < L_{sys}$ LLC slices are assigned.

The on-chip L1 cache, L2 cache, and LLC in the CPU store the copies of frequently used data based on an LRU manner. Each LLC slice can be accessed by at most one CPU core at a time. There is no duplicate cache line throughout the logical LLC to simplify the data coherency among LLC slices. Therefore, requests to the same cache line need to wait until the LLC slice that contains the corresponding cache line finishes serving the previous request.

The table lookup model in the 3D-stacked DRAM was presented in [6], [26]. When a memory request arrives at the 3D-stacked DRAM controller, the hash-function-based distributor classifies the request to one of the queues that correspond to each partial table located in a bank of 3D-stacked DRAM. The banks in the same channel can be interleaved, which enables the multiple partial tables in the same channel to be accessed efficiently. A request entering the queue is served

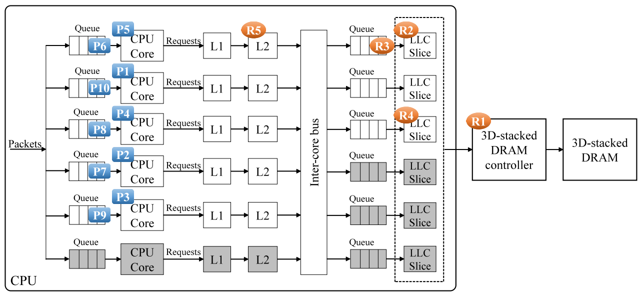


FIGURE 10. Example of system model of proposed architecture where $C_{\mathrm{sys}} = 6$, $L_{\mathrm{sys}} = 6$, $C = 5$, $L = 3$. There are ten packets P1, P2, . . ., P10.

at a channel/bank set in a first come first served (FCFS) policy. The channel/bank set to which a request transferred is selected so that the number of interleaved bank in each channel is minimum.

When a packet comes to the system, a CPU core that processes the packet is randomly selected from the assigned CPU cores. Then, the packet waits at the queue in front of the selected CPU core. At each CPU core, the packet is processed as an FCFS policy. The CPU core issues a memory request to process the packet. Each issued request accesses the L1 cache of the CPU core at first. When there is a miss, the request accesses the L2 cache. The inter-core bus transfers a request that misses the L2 cache to the queue in front of the LLC slice according to the memory address of the request. In this system model, the memory address range that corresponds to the n-th cache line in the memory address space is mapped to the $(n \mod L + 1)$-th LLC Slice. In each LLC slice, a request in the queue in front of the LLC slice is processed as an FCFS policy. A request that misses the LLC is transferred to the 3D-stacked DRAM controller to wait for being distributed to one of channel/bank sets of the 3D-stacked DRAM.

Figure 10 describes an example state of the system model of the proposed architecture. In the example, the system model consists of five assigned CPU cores, three assigned LLC slices, 3D-stacked DRAM, and its controller; this example corresponds to the case presented in Figure 9(b). Each CPU core and LLC slice has its own queue in front of it. There are ten incoming packets in the order of P1, P2, . . ., P10. The first five of them are processed in the five CPU cores, and the other five packets are randomly distributed to the queue of each assigned CPU core regardless of the state of each CPU core. Each CPU core issues a memory request that corresponds to the packet. In other words, the five memory requests R1, R2, . . ., R5 correspond to the first five incoming packets P1, P2, . . ., P5. R1 misses all the levels of cache memories, and then it is in the queue of the 3D-stacked DRAM controller. R2 and R3 are transferred to the topmost LLC slice according to their memory addresses via inter-core bus. R2 is accessing the LLC slice. R3 is waiting in a queue of the same LLC slice as R2. R4 is transferred to the third LLC slice from the top according to its memory address via an inter-core bus. R5 is accessing L2 cache.

A memory request finishes when its corresponding table entry is found in any part of the system. In the system, the total number of requests is limited including all the waiting requests in each queue and the requests that are processed by each CPU core. Let K be the maximum number of requests in the system. A newly issued memory request by a CPU core is blocked when the total number of requests that are in the system equals K. Each queue in front of each CPU core stores the incoming packets to the system. The size of each queue is not less than K, which means that there is no packet loss in each queue.

Let M_{L1}, M_{L2}, M_{Slice}, and M_{3D} denote the memory capacities of L1 cache, L2 cache, LLC slice, and 3D-stacked DRAM, respectively, each of which is given in the system model. The memory sizes of each cache line and each table entry are represented by B, b, respectively. When there is a miss in LLC, B/b table entries are copied to L1 cache from off-chip 3D-stacked DRAM. Then, the LRU cache line in the L1 or L2 cache of the corresponding CPU core is evicted to the L2 cache of the CPU core or LLC slice according to the aforementioned address mapping rule, and the LRU cache line in the LLC slice is dropped.

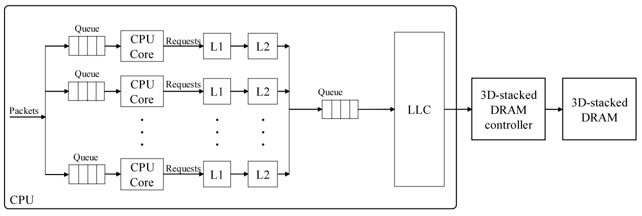
When there is a hit, the cache line that has the requested table entry is allocated in the L1 cache as the most recently used cache line. When there is a miss, the LRU cache line in the L1 or the L2 cache is evicted to the L2 cache or LLC.

B. SYSTEM MODEL OF REFERENCE ARCHITECTURE

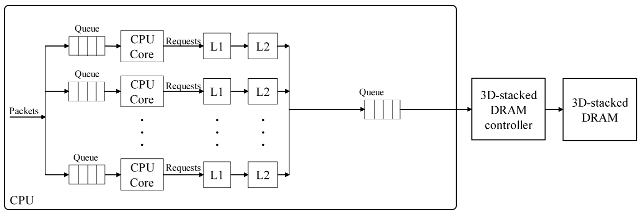
Figure 11 shows the two system models of a reference architecture to be compared with the proposed architecture. The devices that comprise the reference architecture are the same as the proposed architecture.

Figure 11(a) shows the system model of a reference architecture that has a physically shared LLC. The LLC is shared among all the CPU cores in the CPU. The capacity of shared LLC is given and is defined as M_{LLC}. The behavior of this system model is the same as the system model shown in Figure 9, except for how a request that misses L2 cache moves before it hits or misses the LLC. We assume that the shared LLC in this model comprises a single LLC slice whose memory capacity is M_{LLC}, which has a shared queue. A request that misses L2 cache, it waits for the access to the shared LLC in the shared queue in front of the shared LLC.

Figure 12 describes an example state of the system model of reference architecture shown in Figure 11(a). In this example, the system model consists of a CPU equipped with six CPU cores, each of which has a queue in front of it and has dedicated L1 and L2 cache, a shared LLC with a queue in front of it, a 3D-stacked DRAM and its controller. The five of the six CPU cores are assigned for packet processing. As well as the example shown in Figure 10, there are ten incoming packets in the order of P1, P2, . . ., P10. The first five of them are processed in the five assigned CPU cores, and the other five packets wait in each queue in front of the randomly selected CPU core until the processing of the previous packet finishes. Each CPU core issues memory requests that



(a)



(b)

FIGURE 11. System model of reference architecture. (a) Architecture with on-chip shared LLC. (b) Architecture without on-chip LLC.

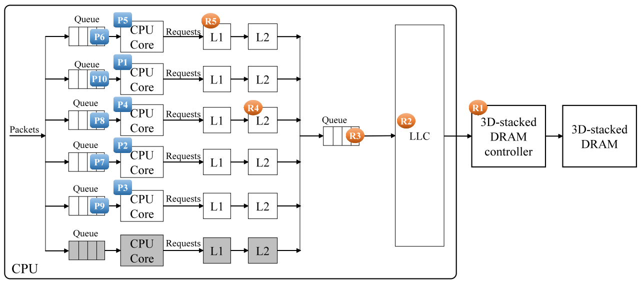


FIGURE 12. Example state of architecture with on-chip shared LLC where $C_{sys} = 6$, $C = 5$. There are ten packets P1, P2, . . ., P10.

correspond to the packet. In other words, the five memory requests R1, R2, . . ., R5 that correspond to the first five incoming packets P1, P2, . . ., P5. R1 misses all the levels of cache memories, and then it is in the queue of the 3D-stacked DRAM controller. R2 and R3 also miss the L2 cache of each CPU core, and then they are transferred to shared LLC. R2 is accessing LLC, and R3 is waiting for access to LLC until LLC finishes serving R2. R4 is accessing the L2 cache, and R5 is accessing the L1 cache. If R4 and R5 miss both the L1 cache and the L2 cache, they will be transferred to shared LLC.

Figure 11(b) shows the system model of reference architecture without on-chip LLC as presented in [24]. All the behavior of this system model is the same as the other system models except for LLC. A request that misses L2 cache is transferred to a common queue and enters the 3D-stacked DRAM controller as an FCFS policy.

For both system models that correspond to Figure 11, the system parameters C_{sys}, C, K, M_{L1}, M_{L2}, M_{3D}, B, and b are common with the description in Section IV-A. When the LRU cache line in the L2 cache is evicted, the cache line is transferred to the shared LLC in the system model that corresponds to Figure 11(a). For the system model that corresponds to Figure 11(b), the LRU cache line in the L2 cache is dropped if the LRU cache line in the L1 cache is evicted to the L2 cache.

V. PERFORMANCE EVALUATION

A. TRAFFIC MODEL

For performance evaluation of the caching mechanism, there are several types of traffic modeling; based on the real traffic traces [14]–[16]; by generating the synthetic traces where the content popularity is taken into account [17], [18]. In [17], [18], a Zipf-like distribution [19] used to model the requests for Web pages and IP table lookup. A Zipf-like distribution can be applied to various situations where a few most popular contents are frequently requested; on the other hand, the rest of the contents are rarely requested.

In the Zipf-like distribution, the parameter α has different values for various traffic traces [19]. It is reported that $\alpha = 1$ is the special case of the Zipf-like distribution, which is known as the strict Zipf's law, does not suit the content distribution such as IP addresses. Typical value of α is in the range of $0 < \alpha < 1$. The value of α is larger for the traffic traces of network with diverse users than that of a homogeneous environment. Thus, when α becomes larger, a few most popular contents are requested more frequently.

In our performance evaluation, we assume that the packet arrival at the system follows a Poisson arrival process whose average rate is λ. We define I as the number of whole table entries accommodated in a channel of 3D-stacked DRAM. The table entries are in order of decreasing popularity. We assume that the requested table entry of each packet follows a Zipf-like distribution. This means that the probability of the ith most popular entry is requested is denoted as $\frac{1}{i^\alpha}$, which is represented as $\frac{\frac{1}{i^\alpha}}{\sum_{i=1}^{I}\frac{1}{i^\alpha}}$ when normalized to constant. Note that, in our performance evaluation, the Poisson and Zipf-like distributions are orthogonal or independent from each other. According to [19], we set $\alpha = 0.83$ in performance evaluation unless otherwise stated.

We assume that each bank of the 3D-stacked DRAM has an equal probability of being accessed when the content popularity is taken into account. We apply the table lookup scheme presented in [6], [26], where requests randomly access banks of 3D-stacked DRAM.

B. SYSTEM ASSUMPTION

In this subsection, we introduce our assumption of the system model for the simplicity of the numerical simulation.

We assume in-order CPU cores and blocking cache memories. Each CPU core processes only one request at a time. Thus the number of issued memory requests at a time is at most the number of CPU cores that the system has, C_{sys}. Thereby we assume that the size of the queue in front of each LLC slice, the shared LLC, and 3D-stacked DRAM controller equals the number of CPU cores, C_{sys}.

We consider that the clock frequency of CPU cores and its associated L1/L2 caches may dynamically change in operation. For instance, Intel's Turbo Boost Technology [70] automatically increases clock frequency of CPU cores for peak loads if the power consumption and temperature of the CPU are below the specification limits. Thus, in the simulations, we assume that the processing times in L1/L2 caches follow exponential distributions. According to [29], the access time to LLC slice varies depending on the distance between a CPU core and an LLC slice. In the simulations, we assume that the processing time of LLC slices follows an exponential distribution. The processing time of 3D-stacked DRAM is considered based on the DRAM mechanism, such as precharge, where the row buffer is written back to the DRAM cell before loading another row.

We also assume that the system is in a stable condition, where there is no memory write requests for such as table update. This assumption allows us to ignore cache coherency. Every data transfer between each level of cache memory and the off-chip 3D-stacked DRAM is performed in 8-byte blocks.

We assume that, in carrier-scale networks, the data that associate with VNFs may not be accommodated in the on-chip caches due to a large number of subscribers. For instance, the virtualized EPC (vEPC) functions and functions of gateway routers or edge routers have multiple internal packet processing functions, each of which has its own data. While we take the table lookup as an example of packet processing, the type of data in such VNFs are not limited to the lookup tables, and there can be other data considered for the VNFs. As a result, some of the data may be accommodated outside of the on-chip caches. To reproduce this situation, the memory capacity of each cache and the number of table entries are considered to be smaller than those of today's COTS hardware in our evaluation. Without this assumption on the memory capacity of each cache and the number of table entries, we cannot finish our numerical simulations in a practical time. For instance, if we set $M_{\mathrm{L1}} = 64$ [KiB], $M_{\mathrm{L2}} = 512$ [KiB], $M_{\mathrm{Slice}} = 1$ [MiB], $M_{\mathrm{LLC}} = 28$ [MiB], the estimated time to obtain a one-plot result is at least one month.

C. BLOCKING PROBABILITY AND AVERAGE WAITING TIME

Let R denote a set of requests that are issued by CPU cores during a certain period of time. The total number of requests in R is represented by $|R|$. A memory request is blocked if the total number of already accommodated requests in the system equals K. Let R_b be the set of blocked requests, where $|R_b|$ denotes the number of blocked requests. Blocking probability is a probability that a newly issued memory request is blocked. Thus, blocking probability is defined by $P_b = \frac{|R_b|}{|R|}$. Throughput, λ_e, is defined by $\lambda_e = \lambda(1 - P_b)$. For accepted request $r \in R \backslash R_b$, we define t_r as the waiting time until it is processed by the CPU core. Thereby, we define W_e as the average effective waiting time by $W_e = \frac{\sum_{r \in R \backslash R_b} t_r}{|R| - |R_b|}$.

We consider the 3D-stacked DRAM that has S channels each of which has two banks. The processing rates of 3D-stacked DRAM channel with and without memory interleaving are $\mu = \mu_1$ and μ_2, respectively. Let ρ be a traffic load, which is defined by $\rho = \frac{\lambda}{S\mu}$. $S\mu$ means the average number of memory requests that a system can process per unit time.

Let N denote the number of table entries stored in the 3D-stacked DRAM. We consider the Internet Protocol (IP) addresses lookup based on DIR-24-8-BASIC [21] algorithm for the benchmark of the packet processing, where the size of each entry is 2 byte. Thereby, we set $b = 2$ [B]. We also set the following values for each system parameter unless otherwise stated, $K = 100$, $\alpha = 0.83$, $M_{\mathrm{L1}} = 128$ [B], $M_{\mathrm{L2}} = 512$ [B], $M_{\mathrm{Slice}} = 146$ [B], $M_{\mathrm{LLC}} = 4096$ [B], $C_{\mathrm{sys}} = 28$, $L_{\mathrm{sys}} = 28$, $\rho = 0.7$, $N = 2 \times 10^4$, $\mu_{\mathrm{L1}} = 100$, $\mu_{\mathrm{L2}} = 50$, $\mu_{\mathrm{LLC}} = 10$, $\mu = \mu_1 = 1$, $\mu_2 = 0.7$, $S = 32$. The memory capacity of 3D-stacked DRAM is considered as $M_{\mathrm{3D}} = 4$ [GiB], which is large enough to store all the entries in the system.

D. NUMERICAL SIMULATION RESULTS

We present the numerical simulation results of three system models, each of which corresponds to the following architecture: (a) proposed architecture with on-chip LLC slices and off-chip 3D-stacked DRAM, (b) architecture with on-chip shared LLC and off-chip 3D-stacked DRAM, and (c) architecture without on-chip shared LLC and with off-chip 3D-stacked DRAM. In this subsection, we label these three as *Proposed*, *With LLC*, and *Without LLC*.

Figures 13, 14, and 15 show the blocking probabilities, average effective waiting times, and throughputs for different numbers of entries in each cache line with different architectures, respectively, where $C = 28$, $L = 28$, $\rho = 0.7$, and $N = 2 \times 10^4$; the set of numbers of entries in each cache line is considered as $\{1, 2, 4, 16, 32, 64\}$. We observe that, as the number of entries in each cache line increases, the blocking probability increases; the average effective waiting time increases; the throughput decreases. This is because that more entries which are unpopular but are stored in the same cache line with some popular entries appear in each cache as the number of entries in each cache line increases. As a result, the capacity of each cache is utilized inefficiently, which decreases the hit probability in each level of cache and degrades the system performances in terms of the blocking probability, the average effective waiting time, and the throughput. We observe that the proposed architecture outperforms the other two architectures in terms of each considered aspect. Today's typical computer systems have 64-byte cache lines, which corresponds to $B/b = 32$. When $B/b = 32$, the proposed architecture reduces the memory access latency by 62 % and 12 % and increases the throughput by 108 % and 2 % with reducing the blocking probability by 96 % and 50 %, compared to the architecture with on-chip shared LLC and that without on-chip LLC, respectively.

These results can be converted to packet processing performance by using the required number of memory accesses of a packet processing application to process a packet. For instance, let the system have a lookup table for IP addresses lookup, where table entry of each IP address is allocated

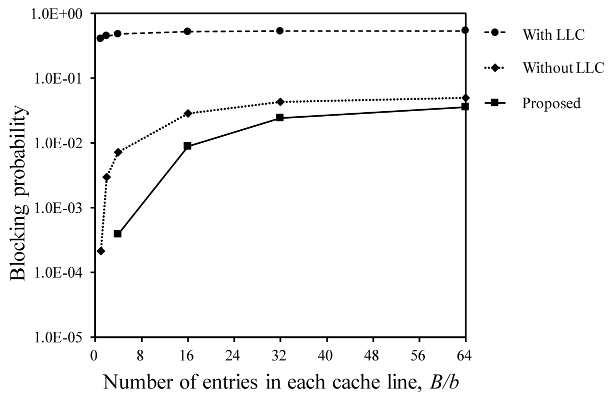


FIGURE 13. Blocking probability depending on number of entries in each cache line.

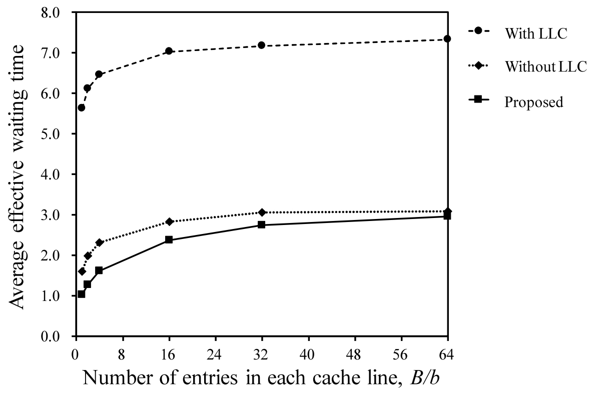


FIGURE 14. Average effective waiting time depending on number of entries in each cache line.

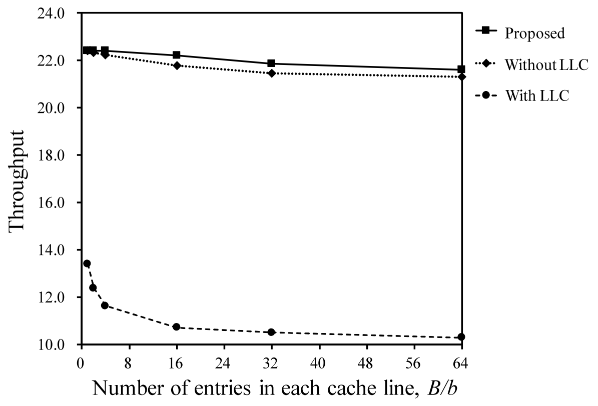


FIGURE 15. Throughput depending on number of entries in each cache line.

in a flat manner so that every lookup for a packet finishes with one memory access. Typical latency of single memory access to HMC is usually between 100-180 [ns] with an average of 125 [ns], according to [68], [69]. Thus we assume that the service rate of the 3D-stacked DRAM, μ, is 8 M services per second. As well as the same configuration for Figures 13, 14, 15, we set $C = 28$, $L = 28$, $S = 32$, $K = 100$, $\rho = 0.7$, $\alpha = 0.83$, $B/b = 32$, and $N = 2 \times 10^4$. Based on these configurations, we calculate the throughput of processing 64 [B] packets and latency of single memory access. The calculated throughput and latency of the proposed

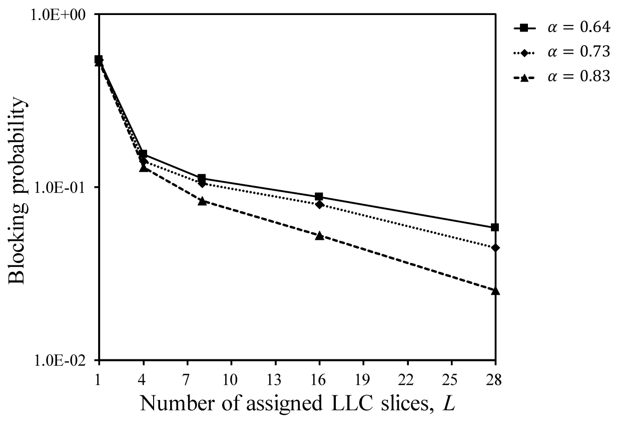


FIGURE 16. Blocking probability depending on number of assigned LLC slices.

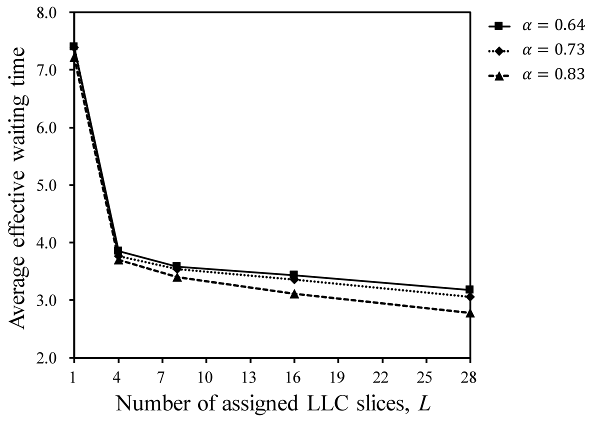


FIGURE 17. Average effective waiting time depending on number of assigned LLC slices.

architecture, the architecture without LLC, and the architecture with LLC are 89.8 [Gbps] and 333 [ns], 87.8 [Gbps] and 381 [ns], and 43.0 [Gbps] and 896 [ns], respectively.

Figures 16, 17, and 18 show the dependencies of blocking probabilities, average effective waiting times, and throughputs in different levels of caches to the number of assigned LLC slices,[1] respectively, with considering three values of α, $C = 28$, $B/b = 32$, $\rho = 0.7$, and $N = 2 \times 10^4$; the set of numbers of assigned LLC slices, L, is considered as $\{1, 4, 8, 16, 28\}$. We observe that, as the number of assigned LLC slices increases, since LLC can process more packets at the same time, which reduces the total time processing each packet, the system performs better in terms of each considered aspect. When L is relatively small compared to C, the system performance increases rapidly as L increases. The on-chip cache memories become more effective for larger α where only a few most popular entries are frequently requested.

Figures 19, 20, and 21 show the blocking probabilities, average effective waiting times, and throughputs for different numbers of assigned CPU cores considering different numbers of assigned LLC slices with different architectures,

[1]Note that the same value of μ_{LLC} is used for different values of L.

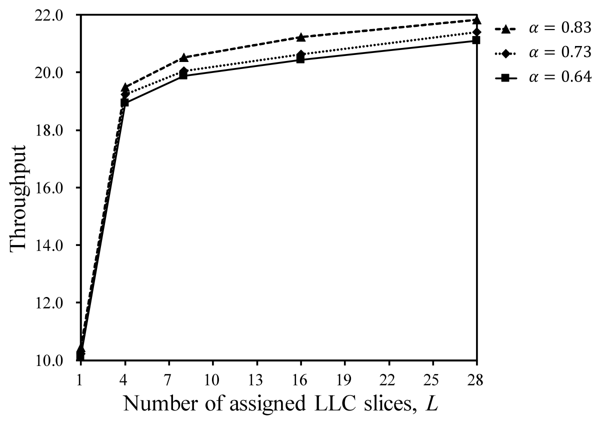


FIGURE 18. Throughput depending on number of assigned LLC slices.

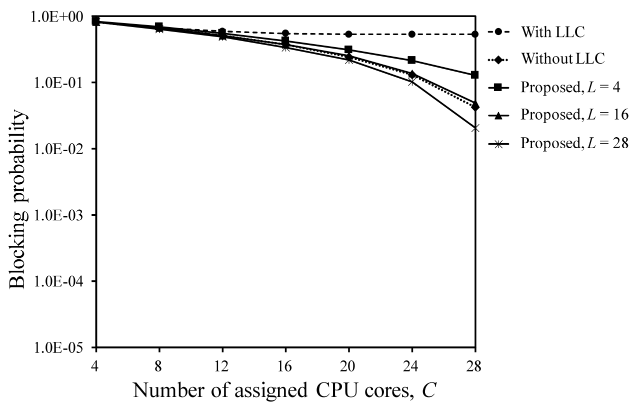


FIGURE 19. Blocking probability depending on number of assigned CPU cores $B/b = 32$.

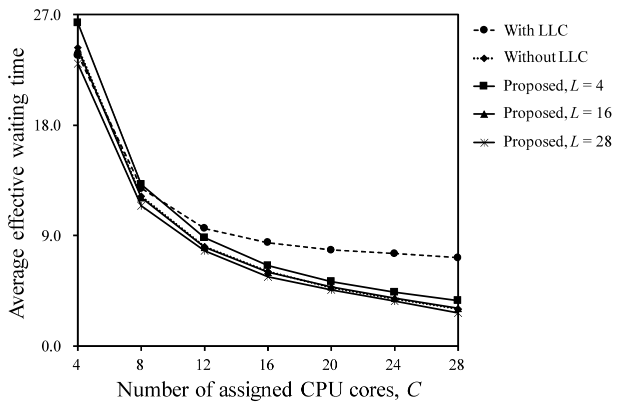


FIGURE 20. Average effective waiting time depending on number of assigned CPU cores $B/b = 32$.

respectively, where $B/b = 32$, $\rho = 0.7$, and $N = 2 \times 10^4$; the set of numbers of assigned CPU cores is considered as $\{4, 8, 12, 16, 20, 24, 28\}$. As the number of assigned CPU cores increases, more requests can be processed at the same time. Consequently, the blocking probability decreases; the average effective waiting time decreases; the throughput increases. As the number of assigned LLC slices increases, the performance of the proposed architecture becomes better. Especially, the architecture without LLC outperforms the proposed architecture with $L = 4$, but when L is set to

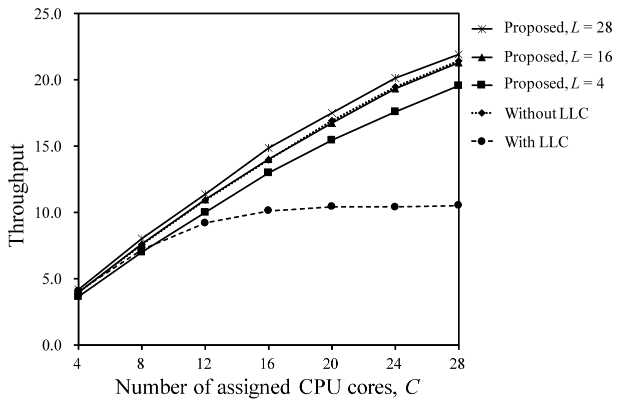


FIGURE 21. Throughput depending on number of assigned CPU cores $B/b = 32$.

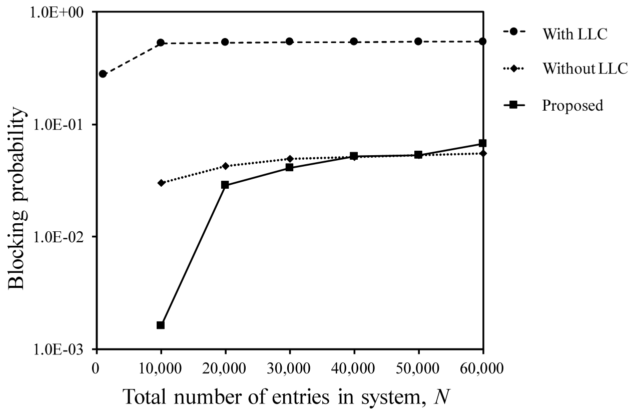


FIGURE 22. Blocking probability depending on total number of entries.

16 and 28, the proposed architecture outperforms the architecture without LLC.

Figures 22, 23, 24, and 25 show the blocking probabilities, average effective waiting times, throughputs, and hit rates of each level of cache for different total numbers of entries in the system with different architectures, respectively, where $C = 28$, $L = 28$, $B/b = 32$, and $\rho = 0.7$; the set of total numbers of entries in the system is considered as $\{10^3, 10^4, 2\times10^4, 3\times 10^4, 4 \times 10^4, 5 \times 10^4, 6 \times 10^4\}$. We observe that, as the total number of entries in the system increases, the blocking probability increases; the average effective waiting time increases; the throughput decreases. This is because, as the total number of entries in the system increases, the proportion of entries stored in the caches decreases, which degrades the efficiency of the cache. As a result, the hit rate in each level of cache decreases, and the system perform worse in terms of each considered aspect. We observe that the increasing speeds of blocking probability and average effective waiting time and the decreasing speed of throughput are slower as the total number of entries increases. When the value of N is small, or $N = 10^4$, the proposed architecture significantly outperforms the architecture without LLC. The benefit of proposed architecture decreases as the value of N increases; when the value of N is greater than one point, the architecture without LLC slightly outperforms the proposed architecture. For a larger total number of entries than a certain point where the

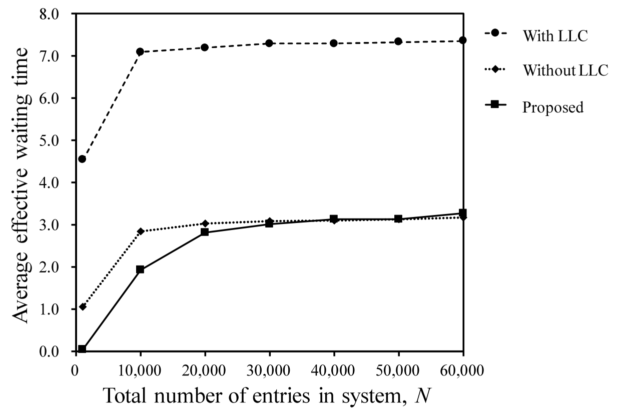


FIGURE 23. Average effective waiting time depending on total number of entries.

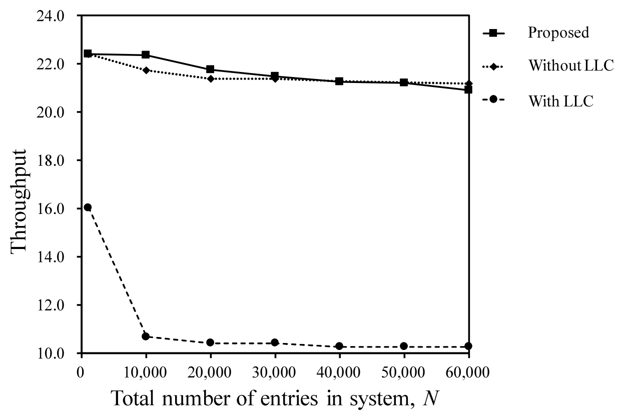


FIGURE 24. Throughput depending on total number of entries.

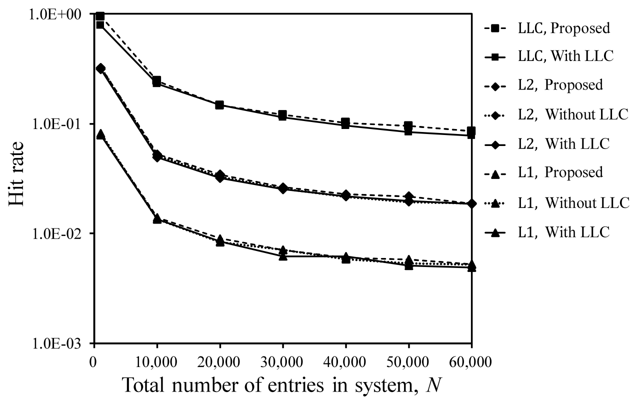


FIGURE 25. Hit rate of each cache level depending on total number of entries.

architecture without LLC begins to outperform the proposed architecture.

For better operation of the proposed architecture when the benefit of on-chip LLC slices decreases, there are several approaches: to skip the on-chip LLC [42], [43], which allows the requests that miss the L2 cache directly access the off-chip 3D-stacked DRAM or to skip the on-chip LLC when allocating unlikely to be reused cache line instead of replacing the LRU content of the LLC for every cache miss [44]–[49]. In the proposed architecture, an application may skip the on-chip LLC slices based on the operator's assignment of LLC slices or the determination of the skip selection logic in the FPGA. By considering the skip of on-chip LLC slices, the proposed architecture will behave similar to the architecture without LLC when the total number of entries is larger than a certain point where the architecture without LLC starts to outperform the proposed architecture. Meanwhile, the released LLC slices may be assigned to other applications in the multi-tenant NFV environment.

VI. RELATED WORK

Several studies presented software-based packet processing on top of COTS hardware systems in [3]–[5], [20]. RouteBricks [3] is the first study that makes the most of the parallelism of modern multi-core CPUs. Lagopus [4] is a high-performance OpenFlow switch software that achieves 10 Gbps switching with 1 M flow entries by using DPDK and on-chip cache memories. These approaches improve packet processing performance compared to the previous packet processing applications running on single-core CPUs. However, in NFV-aware, carrier-scale environment, the capacity of on-chip cache memories are too small to accommodate multiple huge tables, which requires faster, more parallel, and larger memory devices. Graphics Processing Unit (GPU) is used in order to augment the parallelism of packet processing in [20], which is a reasonable extension of COTS hardware. However, the study assumes every packet goes through the same processing in order to leverage the GPU's Single Instruction Multiple Data (SIMD) parallelism. In general, NFV applications process the packets from various users, each of which may have different destination, priority, and packet size, for example. Thus the SIMD-based parallel packet processing may not be applicable to multi-tenant, virtualized carrier networks. Poptrie [5] is a software scheme for high-performance IP routing. It provides 200 M lookups per second performance on a single CPU core. However, this software scheme depends on the small on-chip cache memories. Also, this approach is only for the routing tables and it cannot be applied to other general NFV applications. The proposed architecture improves the overall memory access performance for NFV applications rather than a specific application.

An HMC, a sort of 3D-stacked DRAM device, is used for a packet matching system in [22]. It implements a packet matching circuit in FPGAs, and the FPGA performs table lookups from the HMC. However, there is no detailed discussion on both channel level parallelism and bank interleaving of an HMC. CasHMC [23] provided a cycle-accurate simulator of HMCs. However, it does not cover the bank interleaving of HMCs and is limited to HMC devices itself, which may not be used for combined architecture for CPUs including on-chip cache memories and HMCs.

The thermal feasibility of 3D-stacked DRAM devices was studied in [32]–[34]. For Processing-in-Memory (PIM), the logic layer of HMC can be used to perform logical operations to the data in the DRAM layers. Although PIM reduces the latency of data transfer between the processor and the HMC, PIM also produces much heat, which requires powerful cooling. According to the studies above, PIM use cases using 3D-stacked DRAM are feasible if the system has high-end active cooling. Thereby, the proposed architecture is feasible since it uses 3D-stacked DRAM for simple memory accesses without depending on the functions of the logic layer of 3D-stacked DRAM, which enables us to use the commodity coolers of COTS systems.

3D die-stacked DRAM cache architectures were studied in [37]–[40]. These approaches directly stack DRAM cache memories on the CPU, which lowers access latency to the DRAM cache memories compared to the off-chip DRAMs while 3D die-stacked DRAM cache extends the capacity of cache memories instead of using small on-chip SRAM-based cache memories. Although there are some GPU products or co-processor products that use 3D-stacked DRAM, applying the 3D die-stacked DRAM cache to the mainstream server CPUs such as Intel Xeon needs significant modification of chip's microarchitecture.

In terms of utilizing LLC slices, data replication among the LLC slices was introduced to increase the effective memory access bandwidth in parallel processing in [50]. It improves the processing speed and the energy efficiency based on the LLC slices. Due to the limited memory capacity of the on-chip LLC slices, the data replication technique is not applicable to the packet processing that requires larger memory space.

The methods to improve the packet processing performance in the NFV-aware environment were introduced in [63], [64]. The work in [63] showed that the number of memory copies and the number of accesses to remote processors in non-uniform memory access (NUMA) environments significantly affect the packet processing performance. The work in [63] also demonstrated that larger table data size degrades the packet processing performance due to an increasing number of cache misses as the table data size increases, which is compatible with the motivation of the introduction of the 3D-stacked DRAM. It also presented the assignment of resources such as packet queues in NIC hardware and CPU cores to VNFs should be optimized. The work in [64] introduced smart NICs to COTS hardware for NFV. A smart NIC is the NIC that has a processor or an FPGA so that some part of packet processing in NFV applications can be offloaded to the NIC. In this study, the Distributed Denial of Service (DDoS) mitigation function is offloaded to a smart NIC in order to improve the packet processing performance at the CPU by releasing the CPU from DDoS mitigation processing. These approaches are compatible with the proposed architecture, where the proposed architecture improves the overall memory access performance.

VII. DISCUSSION

We assume that, for the initial understanding of the performance dependency on a part of the system parameters before we determine the detailed design of the system architecture, there is an approach to observe the system performance based on the simplified model in terms of the focused parameters and features of the system, instead of the detailed, full-system model and its simulator with lots of system parameters, complicated parameter tuning for the simulations, and the various hardware/software features. Then, based on the initial understanding, we further incorporate the other system parameters and the hardware/software features toward the understanding of a more realistic system.

We firstly focus on the hardware aspect of the memory parallelism based on the motivation of this work, which is presented in Section I. We modeled the proposed architecture in terms of the memory parallelism of the LLC slices and the 3D-stacked DRAM. Although the system models in this work do not incorporate the details of the actual hardware/software techniques such as out-of-order CPUs and memory address translation in the Operating System (OS), the simplified system model allows us to observe the initial performance dependency on the number of assigned resources such as LLC slices.

We plan to use PIM-related features of the 3D-stacked DRAM in our future work, which is not used in the current proposed architecture, to increase the performance and the power efficiency of the system by reducing the number of data/request transfers between the CPU and the 3D-stacked DRAMs. In such a further discussion on the packet processing architecture that uses the PIM technique, for instance, the details of the actual hardware/software and their corresponding parameters, including the capacity of each cache, will be worth incorporated in our future study.

VIII. CONCLUSION

This paper proposed a packet processing architecture that uses both on-chip LLC slices and off-chip interleaved 3D-stacked DRAM devices for parallel packet processing. Table entries are stored in the off-chip 3D-stacked DRAM so that memory requests are processed in parallel by using bank interleaving and channel parallelism. Also, cache entries are distributed according to a memory address-based hash function so that each CPU core accesses on-chip LLC in parallel. We evaluated the memory access performance of the proposed architecture in terms of average effective waiting times, throughputs, and blocking probabilities with considering several system parameters. The evaluation results showed that the proposed architecture reduces the memory access latency by 62 % and 12 % and increases the throughput by 108 % and 2 % with reducing blocking probability 96 % and 50 %, compared to the architecture with on-chip shared LLC and that without on-chip LLC, respectively. Also, with a larger number of assigned LLC slices, the performance of proposed architecture increases, while the number of assigned LLC slices does not significantly affect the system performance if more than half of the total number of LLC slices are assigned.

REFERENCES

[1] *Intel Data Plane Development Kit*. Accessed: Jul. 28, 2019. [Online]. Available: http://dpdk.org/

[2] *PCI-SIG Single Root I/O Virtualization (SR-IOV) Support in Intel Virtualization Technology for Connectivity*. Accessed: Jul. 28, 2019. [Online]. Available: https://www.intel.com

[3] M. Dobrescu, N. Egi, K. Argyraki, B. G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," in *Proc. ACM SIGOPS*, 2009, pp. 15–28.

[4] *Lagopus Switch, a High Performance Software OpenFlow 1.3 Switch*. Accessed: Jul. 28, 2019. [Online]. Available: http://www.lagopus.org/

[5] H. Asai and Y. Ohara, "Poptrie: A compressed trie with population count for fast and scalable software IP routing table lookup," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 57–70, Aug. 2015.

[6] T. Korikawa, A. Kawabata, F. He, and E. Oki, "Carrier-scale packet processing system using interleaved 3D-stacked DRAM," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[7] N. Oliver, R. R. Sharma, S. Chang, B. Chitlur, E. Garcia, J. Grecco, A. Grier, N. Ijih, Y. Liu, P. Marolia, H. Mitchel, S. Subhaschandra, A. Sheiman, T. Whisonant, and P. Gupta, "A reconfigurable computing system based on a cache-coherent fabric," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Nov. 2011, pp. 80–85.

[8] Y. Watanabe, Y. Kobayashi, T. Takenaka, T. Hosomi, and Y. Nakamura, "Accelerating NFV application using CPU-FPGA tightly coupled architecture," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Dec. 2017, pp. 136–143.

[9] D. J. Moss, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "A customizable matrix multiplication framework for the intel HARPv2 Xeon+FPGA platform: A deep learning case study," in *Proc. ACM/SIGDA FPGA*, 2018, pp. 107–116.

[10] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic-performance characteristics of system-on-chip communication architectures," in *Proc. 14th Int. Conf. VLSI Design*, 2001, pp. 29–35.

[11] I. Y. Bucher and D. A. Calahan, "Models of access delays in multiprocessor memories," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 3, pp. 270–280, May 1992.

[12] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for MPEG-2 video applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 1, pp. 108–119, Jan. 2004.

[13] V. Soteriou, H. Wang, and L. Peh, "A statistical traffic model for on-chip interconnection networks," in *Proc. 14th IEEE Int. Symp. Modeling, Anal., Simulation*, Sep. 2006, pp. 104–106.

[14] E. Cohen and H. Kaplan, "Proactive caching of DNS records: Addressing a performance bottleneck," *Comput. Netw.*, vol. 41, no. 6, pp. 707–726, Apr. 2003.

[15] T. Chiueh and P. Pradhan, "High-performance IP routing table lookup using CPU caching," in *Proc. IEEE INFOCOM Conf. Comput. Commun., 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. Future Now*, Mar. 2003, vol. 41, no. 6, pp. 707–726.

[16] S. Ihm and V. S. Pai, "Towards understanding modern Web traffic," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2011, pp. 295–312.

[17] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 589–603, Oct. 2002.

[18] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "DiCAS: An efficient distributed caching mechanism for P2P systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 10, pp. 1097–1109, Oct. 2006.

[19] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM Conf. Comput. Commun., 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. Future Now*, vol. 1, Mar. 1999, pp. 126–134.

[20] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, Aug. 2010.

[21] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. IEEE INFOCOM*, vol. 3, Mar. 1998, pp. 1240–1247.

[22] D. Rozhko, G. Elliott, D. Ly-Ma, P. Chow, and H.-A. Jacobsen, "Packet matching on FPGAs using HMC memory: Towards one million rules," in *Proc. ACM/SIGDA FPGA*, 2017, pp. 201–206.

[23] D.-I. Jeon and K.-S. Chung, "CasHMC: A cycle-accurate simulator for hybrid memory cube," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 10–13, Jan. 2017.

[24] T. Korikawa, A. Kawabata, F. He, and E. Oki, "Packet processing architecture with off-chip LLC using interleaved 3D-stacked DRAM," in *Proc. IEEE 20th Int. Conf. High Perform. Switching Routing (HPSR)*, May 2019, pp. 1–6.

[25] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "HBM (high bandwidth memory) DRAM technology and architecture," in *Proc. IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, May 2017, pp. 1–4.

[26] T. Korikawa, A. Kawabata, F. He, and E. Oki, "Carrier-scale packet processing architecture using interleaved 3D-stacked DRAM and its analysis," *IEEE Access*, vol. 7, pp. 75500–75514, 2019.

[27] A. Kumar. *The New Intel Xeon Scalable Processor (Formerly Skylake-SP)*. Accessed: Jun. 21, 2019. [Online]. Available: https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.90-Server-Pub/HC29.22.930-Xeon-Skylake-sp-Kumar-Intel.pdf

[28] *Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 V4 Family*. Accessed: Jun. 21, 2019. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[29] A. Farshin, A. Roozbeh, G. Q. Maguire, and D. Kostić, "Make the most out of last level cache in Intel processors," in *Proc. 14th EuroSys Conf. (EuroSys)*, New York, NY, USA, 2019, Art. no. 8.

[30] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Systematic reverse engineering of cache slice selection in intel processors," in *Proc. Euromicro Conf. Digit. Syst. Design*, Washington, DC, USA, Aug. 2015, pp. 629–636.

[31] W. Chen, S.-L. Chen, S. Chiu, R. Ganesan, V. Lukka, W. W. Mar, and S. Rusu, "A 22 nm 2.5 MB slice on-die L3 cache for the next generation Xeon processor," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, 2013, pp. C132–C133.

[32] M. J. Khurshid and M. Lipasti, "Data compression for thermal mitigation in the hybrid memory cube," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 185–192.

[33] Y. Zhu, B. Wang, D. Li, and J. Zhao, "Integrated thermal analysis for processing in die-stacking memory," in *Proc. 2nd Int. Symp. Memory Syst. (MEMSYS)*, 2016, pp. 402–414.

[34] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," in *Proc. 47th IEEE/ACM Int. Symp. Microarchitecture (MICRO-47)*, 2014. [Online]. Available: https://www.cs.utah.edu/wondp/eckert.pdf

[35] S. Han, H. Seo, B. Kim, and E.-Y. Chung, "PIM architecture exploration for HMC," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Jeju, South Korea, Oct. 2016, pp. 635–636.

[36] D.-I. Jeon, K.-B. Park, and K.-S. Chung, "HMC-MAC: Processing-in memory architecture for multiply-accumulate operations with hybrid memory cube," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 5–8, Jan./Jun. 2018.

[37] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Cambridge, U.K., Dec. 2014, pp. 25–37.

[38] D. Kim, S. Yoo, and S. Lee, "Hybrid main memory for high bandwidth multi-core system," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 1, no. 3, pp. 138–149, Jul. 2015.

[39] A. Shahab, M. Zhu, A. Margaritov, and B. Grot, "Farewell my shared LLC! A case for private die-stacked DRAM caches for servers," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Fukuoka, Japan, Oct. 2018, pp. 559–572.

[40] Y. Guo, Q. Liu, W. Xiao, P. Huang, N. Podhorszki, S. Klasky, and X. He, "SELF: A high performance and bandwidth efficient approach to exploiting die-stacked DRAM as part of memory," in *Proc. IEEE 25th Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst. (MASCOTS)*, Banff, AB, Canada, Sep. 2017, pp. 187–197.

[41] *Hybrid Memory Cube Specification 2.1*. Accessed: Jun. 21, 2019. [Online]. Available: http://hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf

[42] T. S. Warrier, M. Mutyam, and K. Raghavendra, "SkipCache: Application aware cache management for chip multi-processors," *IET Comput. Digit. Techn.*, vol. 9, no. 6, pp. 293–299, Nov. 2015.

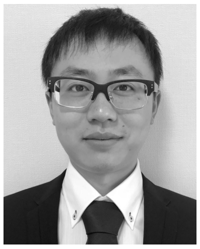
[43] J. Kong and K. Lee, "A DVFS-aware cache bypassing technique for multiple clock domain mobile SoCs," *IEICE Electron. Express*, vol. 14, no. 11, pp. 1–12, 2017.

[44] Y. Tian, S. Puthoor, J. L. Greathouse, B. M. Beckmann, and D. A. Jiménez, "Adaptive GPU cache bypassing," in *Proc. 8th Workshop Gen. Purpose Process. Using GPUs (GPGPU)*, New York, NY, USA, 2015, pp. 25–35.

[45] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Apr. 2008.
[46] S. Gupta, H. Gao, and H. Zhou, "Adaptive cache bypassing for inclusive last level caches," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, Boston, MA, USA, May 2013, pp. 1243–1253.
[47] Y. Huangfu and W. Zhang, "Hardware-based and hybrid l1 data cache bypassing to improve GPU performance," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun., 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, New York, NY, USA, Aug. 2015, pp. 972–976.
[48] G. Sun, C. Zhang, P. Li, T. Wang, and Y. Chen, "Statistical cache bypassing for non-volatile memory," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3427–3440, Nov. 2016.
[49] S. Mittal, "A survey of cache bypassing techniques," *J. Low Power Electron. Appl.*, vol. 6, no. 2, p. 5, 2016.
[50] G. Kurian, S. Devadas, and O. Khan, "Locality-aware data replication in the last-level cache," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Orlando, FL, USA, Feb. 2014, pp. 1–12.
[51] M. H. Hajkazemi, M. K. Tavana, T. Mohsenin, and H. Homayoun, "Heterogeneous HMC+DDRx memory management for performance-temperature tradeoffs," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 1, pp. 1–21, Sep. 2017.
[52] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Portland, OR, USA, Jun. 2015, pp. 336–348.
[53] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for CMP scaling," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2009, pp. 371–382.
[54] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, ACM, New York, NY, USA, 2009, pp. 267–278.
[55] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *Proc. IEEE Int. Symp. High-Perform. Comp Archit.*, New Orleans, LA, USA, 2012, pp. 1–12.
[56] G. Lee, D. Min, I. Byun, and J. Kim, "Cryogenic computer architecture modeling with memory-side case studies," in *Proc. 46th Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, pp. 774–787.
[57] *Hybrid Memory Cube Controller IP Core User Guide*. Accessed: Jan. 13, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/nik1412377950681.html
[58] *Xilinx HMC Controller PG216—XHMC V1.0 Product Guide (V1.0)*. Accessed: Jan. 13, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xhmc/v1_0/pg216-xhmc.pdf
[59] *High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide*. Accessed: Jan. 13, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-20031.pdf
[60] *High Bandwidth Memory (HBM2) Controller and PHY*. Accessed: Jan. 13, 2020. [Online]. Available: http://www.open-silicon.com/high-bandwidth-memory-ip/
[61] *AXI High Bandwidth Memory Controller V1.0*. Accessed: Jan. 13, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf
[62] D. Park, A. Vaidya, A. Kumar, and M. Azimi, "MoDe-X: Microarchitecture of a layout-aware modular decoupled crossbar for on-chip interconnects," *IEEE Trans. Comput.*, vol. 63, no. 3, pp. 622–636, Mar. 2014.
[63] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma, "Towards optimal adaptation of NFV packet processing to modern CPU memory architectures," in *Proc. 2nd Workshop Cloud-Assist. Netw. (CAN)*, New York, NY, USA, 2017, pp. 7–12.
[64] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese, "Introducing SmartNICs in server-based data plane processing: The DDoS mitigation use case," *IEEE Access*, vol. 7, pp. 107161–107170, 2019.
[65] *New Intel Mesh Architecture: The 'Superhighway' of the Data Center*. Accessed: Jan. 19, 2020. [Online]. Available: https://silix.com.br/pdf/Intel/Intel_Mesh_Whitepaper.pdf?0f3648&0f3648
[66] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Los Angeles, CA, USA, Jun. 2018, pp. 383–396.
[67] M. Huang, M. Mehalel, R. Arvapalli, and S. He, "An energy efficient 32-nm 20-mb shared on-die L3 cache for Intel Xeon processor E5 family," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1954–1962, Aug. 2013.
[68] R. Hadidi, B. Asgari, B. A. Mudassar, S. Mukhopadhyay, S. Yalamanchili, and H. Kim, "Demystifying the characteristics of 3D-stacked memories: A case study for hybrid memory cube," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Seattle, WA, USA, Oct. 2017, pp. 66–75.
[69] R. Hadidi, B. Asgari, J. Young, B. Ahmad Mudassar, K. Garg, T. Krishna, and H. Kim, "Performance implications of NoCs on 3D-stacked memories: Insights from the hybrid memory cube," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Belfast, U.K., Apr. 2018, pp. 99–108.
[70] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar./Apr. 2012.
[71] *Intel STRATIX 10 MX FPGAS*. Accessed: Feb. 23, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/sip/stratix-10-mx.html

TOMOHIRO KORIKAWA received the B.S. and M.S. degrees from Waseda University, Tokyo, Japan, in 2012 and 2014, respectively. In 2014, he joined Nippon Telegraph and Telephone Corporation (NTT), Tokyo, where he is researching network system architecture and network design. He is currently a Researcher with the Network Service Systems Laboratories, NTT.

AKIO KAWABATA received the B.E., M.E., and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 1991, 1993, and 2016, respectively. In 1993, he joined Nippon Telegraph and Telephone Corporation (NTT) Communication Switching Laboratories, Tokyo, where he has been engaging to develop switching systems, and researching network design and switching system architecture. He is currently an Executive Research Engineer, and also the Project Manager of Network Service Systems Laboratories, NTT. He is also associated with the Department of Communication Engineering and Informatics, The University of Electro-Communications, Tokyo, for research activities. He has served as the Senior Manager of the R&D Department at NTT East, from 2011 to 2014.



FUJUN HE (Student Member, IEEE) received the B.E. and M.E. degrees from the University of Electronic Science and Technology of China, Chengdu, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with Kyoto University, Kyoto, Japan. He was an Exchange Student at The University of Electro-Communications, Tokyo, Japan, from 2015 to 2016. His research interests include modeling, algorithm, optimization, resource allocation, survivability, and optical networks.

EIJI OKI (Fellow, IEEE) received the B.E. and M.E. degrees in instrumentation engineering and the Ph.D. degree in electrical engineering from Keio University, Yokohama, Japan, in 1991, 1993, and 1999, respectively. He was with Nippon Telegraph and Telephone Corporation (NTT) Laboratories, Tokyo, from 1993 to 2008, and The University of Electro-Communications, Tokyo, from 2008 to 2017. From 2000 to 2001, he was a Visiting Scholar at the Polytechnic Institute of New York University, Brooklyn, NY, USA. In 2017, he joined Kyoto University, Japan, where he is currently a Professor. His research interests include routing, switching, protocols, optimization, and traffic engineering in communication and information networks.