Doctoral Thesis

Parallel Memory System Architectures for Packet Processing in Network Virtualization

Tomohiro KORIKAWA

Graduate School of Informatics, Kyoto University

March 2021

Preface

Network virtualization aims to reduce the capital expenditure (CAPEX) and the operating expenditure (OPEX) of network infrastructure by leveraging commercial off-the-shelf (COTS) hardware such as general-purpose computers and virtual network functions (VNFs) instead of conventional dedicated network equipment. Network functions comprise various packet processing tasks such as classification, table lookup for searching, packet modification, and queueing, each of which issues memory accesses and requires high memory performance. Conventional network equipment comprises purpose-built dedicated components such as processors, memory devices, and bus architecture to satisfy the specifications and requirements of network services. In particular, large-scale service providers such as telecom operators have depended on such conventional equipment to satisfy the service level agreement (SLA) and to accommodate various and a large amount of traffic from subscribers, which prevents the service providers from benefiting from network virtualization. This thesis studies four problems about parallel memory system architectures for packet processing in network virtualization. Each problem corresponds to the main memory parallelism, integration of on-chip cache memories of the CPU with the parallel main memory, capacity and parallelism of the on-chip cache memories in the presence of parallel main memory, and accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses, respectively.

Firstly, this thesis proposes a memory system architecture that uses a three-dimensional (3D)-stacked memory to increase the main memory parallelism for packet processing in network virtualization. In current general-purpose computers such as servers based on x86 central processing unit (CPU), while the overall performance is increasing due to the increasing number of CPU cores, the main memory parallelism is much less than the number of CPU cores, which limits packet processing performance in network virtualization. This work augments main memory parallelism by leveraging both channel-level parallelism and bank interleaving of a 3D-stacked dynamic random access memory (DRAM). In the 3D-stacked DRAM, a database for packet processing, such as a lookup table, is split into partial databases, each of which is allocated to each set of memory channel and bank. A hash-function-based distributor distributes incoming memory requests to an appropriate memory channel-bank set that has the corresponding partial database for the requests. This work introduces an analytical model of the proposed memory system architecture for two traffic patterns, one with random memory request arrivals and one with bursty arrivals. The numerical results observe that the proposed memory system architecture increases packet processing performance up to around 80 Gbps for the smallest-sized Internet Protocol (IP) packets involving random and bursty memory request arrivals.

Secondly, this thesis proposes a memory system architecture that integrates on-chip private cache memories with the off-chip 3D-stacked memory to reduce memory access latency in the existence of main memory parallelism for packet processing in network virtualization. In general-purpose computers, CPUs usually have several levels of on-chip cache memories to obscure the main memory latency. The on-chip cache memories comprise private cache memories that belong to each CPU core and the last-level-cache (LLC) that is shared among all the CPU cores. The proposed architecture integrates the private cache memories of each CPU core with the 3D-stacked DRAM-based main memory. This work explores the memory system architecture in terms of integrating the on-chip cache memories with the off-chip 3D-stacked DRAM, with considering two reference architectures, one with the on-chip private cache memories and the on-chip shared LLC and one without any on-chip cache memories. The evaluation results observe that the proposed memory system architecture reduces memory access latency by 58 % and 1.8 % and increases throughput by 104 % and 1 % with reducing the blocking probability by 91 % and 18 %, compared to the reference architectures with the on-chip private cache memories and the on-chip shared LLC and that without any on-chip cache memories, respectively.

Thirdly, this thesis proposes a memory system architecture that uses the 3D-stacked memory, the on-chip private cache memories, and on-chip LLC slices to increase capacity and parallelism of the on-chip cache memories in the integration with the off-chip parallel main memory for packet processing in network virtualization. The on-chip shared LLC in the latest CPU comprises multiple LLC slices, each of which belongs to one of the CPU cores and can be accessed from each CPU core via a mesh or ring bus. A system operator can assign some of the LLC slices to a specific packet processing task. The proposed architecture integrates the LLC slices with the on-chip private cache memories and the off-chip 3D-stacked DRAM. In the proposed architecture, the cached data is distributed to each LLC slices according to a memory address-based hash function so that CPU cores can access the LLC slices in parallel. This work analyzes the memory performance dependency on the number of assigned LLC slices when integrated with the off-chip 3D-stacked DRAM-based parallel main memory and the on-chip private cache memories that belong to each CPU core. The evaluation results observe that the proposed architecture reduces memory access latency by 62 % and 12 % and increases throughput by 108 % and 2 % with reducing the blocking probabilities by 96 % and 50 %, compared to the reference architectures with the on-chip private cache memories and the on-chip shared LLC and that with the on-chip private cache memories and without the on-chip shared LLC, respectively.

Fourthly, this thesis proposes a memory system architecture that uses a network of 3D-stacked memories to increase throughput and reduce accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses. Packets that enter the memory network receive packet processing at each 3D-stacked DRAM without going back and forth between the memories and the processors until the packet processing is completed. Each 3D-stacked DRAM has several DRAM layers on top of a logic die. The logic die consists of a logic for memory accesses and device interfaces and a user-defined programmable logic, in which each packet processing task is placed. If a packet processing task needs to search a database, the database is stored in the same 3D-stacked DRAM as the task is allocated. The database is replicated and allocated in every memory channel to leverage memory parallelism of 3D-stacked DRAMs. In each

3D-stacked DRAM, the user-defined logic for a packet processing task issues memory requests to the memory channels. For a packet processing task that requires more memory accesses than the other tasks, additional 3D-stacked DRAMs are assigned for the task. The evaluation results observe that the proposed architecture reduces the blocking probability by issuing the next memory request inside the 3D-stacked DRAM just after the previous memory access, instead of allowing the arrivals of incoming packets during the data transfers between the memory and processor. Consequently, the proposed architecture increases throughput and reduces the accumulated latency when there are multiple packet processing tasks, compared to the architecture with 3D-stacked DRAM-based parallel main memory, where every memory access requires data transfers between the processors and memories. The proposed architecture also reduces the blocking probability and latency by assigning more 3D-stacked DRAMs for a packet processing task that requires more memory accesses than the other tasks.

This thesis is organized as follows. Chapter 1 introduces the background of packet processing, computer architectures of dedicated equipment and general-purpose computers, and major hardware devices in computers. Chapter 2 describes related works. Chapter 3 presents the parallel memory system architecture using the interleaved 3D-stacked memory. Chapter 4 presents the parallel memory system architecture using the interleaved 3D-stacked memory and the on-chip private cache memories. Chapter 5 presents the parallel memory system architecture using the interleaved 3D-stacked memory, the on-chip private cache memories, and the on-chip LLC slices. Chapter 6 presents the memory system architecture using the network of interleaved 3D-stacked memories. Finally, chapter 7 concludes this thesis.

Acknowledgements

This thesis is the summary of my doctoral study at Kyoto University. Many people have helped me to do my researches and accomplish this thesis. I would like to take the opportunity to thank all of them.

Fist and foremost, I would like to express my sincere gratitude to Professor Eiji Oki, for his constant guidance and encouragement. Without the critical suggestions and constructive comments from Professor Oki, I could not complete any of my researches nor this thesis. I also received guidance from Professor Oki for doing my future researches as a professional researcher.

I would like to offer my special thanks to Dr. Fujun He, for his enormous contribution to the performance evaluation of the proposed architectures in Sections 3, 4, and 5 of this thesis. Without the models and simulators that Dr. He has constructed, I could not complete the works.

I would particularly like to thank my previous supervisor at Nippon Telegraph and Telephone Corporation, Dr. Akio Kawabata, who gave me constant guidance since I joined the company and encouraged me to obtain a Ph.D. I would like to thank my colleagues at the company, who gave me warm encouragements during my doctoral study.

I would like to thank Professor Masahiro Morikura and Professor Yasuo Okabe for being part of my judging committee and providing their insightful comments to improve this thesis.

I would also like to thank Professor Ryoichi Shinkuma, Professor Takehiro Sato, Ms. Mariko Tatsumi, and all members in Oki lab, who gave me warm welcome and kind helps.

Last but not least, my heartfelt appreciation goes to my parents and sister, who have taken care of my health and warmly supported both my doctoral study and work at the company.

Contents

Preface iii

Acknowledgements vii

List of Figures xvi

List of Tables xvii

Notations xix

Abbreviations xxi

1 Introduction 1

1.1 Packet processing . . . 1

1.1.1 Packet processing types . . . 1

1.1.2 Framing . . . 2

1.1.3 Parsing and classification . . . 3

1.1.4 Searching and forwarding . . . 4

1.1.5 Modification . . . 5

1.1.6 Encryption . . . 5

1.1.7 Queueing and traffic management . . . 6

1.2 Network virtualization . . . 6

1.2.1 Software defined networking (SDN) . . . 7

1.2.2 Network functions virtualization (NFV) . . . 9

1.3 Computer architectures for packet processing . . . 11

1.3.1 Dedicated architecture . . . 12

1.3.2 General-purpose architecture . . . 14

1.4 Processor types in computer systems 16

1.4.1 Application specific integrated circuit (ASIC) 16

1.4.2 Programmable logic device (PLD) 17

1.4.3 Network processor (NP) 19

1.4.4 Central processing unit (CPU) 21

1.5 Memory types in computer systems 22

1.5.1 Static random access memory (SRAM) 22

1.5.2 Dynamic random access memory (DRAM) 24

1.5.3 Emerging memory types 25

1.6 Content addressable memory (CAM) 26

1.7 Memory systems in general-purpose computers 29

1.7.1 Main memory system 29

1.7.2 Cache memory system 32

1.7.3 Three-dimensional (3D)-stacked DRAM 35

1.8 Challenges of packet processing in network virtualization 38

1.9 Problem statements 39

1.9.1 Memory parallelism in main memory 40

1.9.2 Integration of on-chip cache memories with parallel main memory 40

1.9.3 Capacity and parallelism of on-chip cache memories in presence of parallel main memory 41

1.9.4 Accumulated latency of data transfers between processors and memories 42

1.10 Overview and contributions of this thesis 43

2 Related works 47

2.1 Packet processing technologies in network virtualization 47

2.2 3D die-stacked memory architectures 49

2.3 Emerging computer architectures 50

2.4 Computer architecture simulation 53

3 Parallel memory system architecture using interleaved 3D-stacked memory 57

3.1 Proposed architecture 58

3.2 System model 62
3.3 Analysis for random arrival of memory requests 65
3.3.1 States description 65
3.3.2 State transition for (i, j, p) 69
3.3.3 Equilibrium states 69
3.3.4 Blocking probability and average waiting time 71
3.4 Analysis for bursty arrival of memory requests 72
3.4.1 Overview of IPP 72
3.4.2 State transition for (i, j, p, k) 73
3.4.3 Equilibrium states 76
3.4.4 Blocking probability and average waiting time 77
3.5 Evaluation 78
3.5.1 Numerical results for random arrival of memory requests 78
3.5.2 Numerical results for bursty arrival of memory requests . 80
3.5.3 Packet Processing Performance 83
3.6 Direction to expansion of analytical model for general $N \geq 2$ . . 86
3.7 Chapter summary 87

4 Parallel memory system architecture using interleaved 3D-stacked memory and private cache memory 89

4.1 Proposed architecture 90
4.2 System model 92
4.2.1 System model of proposed architecture 92
4.2.2 System models of reference architectures 94
4.3 Evaluation 95
4.3.1 Traffic model 95
4.3.2 System assumption 96
4.3.3 Blocking probability and average waiting time 98
4.3.4 Numerical results 98
4.4 Chapter summary 102

5 Parallel memory system architecture using interleaved 3D-stacked memory, private cache memory, and LLC slices 103

5.1 Proposed architecture 104

5.2 System model 106

5.2.1 System model of proposed architecture 106

5.2.2 System model of reference architectures 110

5.3 Evaluation 113

5.3.1 Traffic model 113

5.3.2 System assumption 113

5.3.3 Blocking probability and average waiting time 114

5.3.4 Numerical results 115

5.4 Chapter summary 125

6 Parallel memory system architecture using network of interleaved 3D-stacked memories 127

6.1 Proposed architecture 128

6.2 System model 131

6.2.1 System model of memory network 131

6.2.2 System model of 3D-stacked DRAM 133

6.3 Numerical results 136

6.3.1 Performance metrics and evaluation settings 136

6.3.2 Performance dependency on number of packet processing tasks and traffic load 137

6.3.3 Performance dependency on number of vertically cascaded 3D-stacked DRAMs and traffic load 141

6.4 Chapter summary 144

7 Conclusions 145

Bibliography 151

Publication 173

List of Figures

1.1 NFV reference architecture. 10
1.2 High-level computer architecture of conventional network equipment. 12
1.3 High-level system architecture of a modular network equipment. 13
1.4 High-level architecture of general-purpose computers based on x86 CPU. 15
1.5 High-level architecture of switch ASIC. 17
1.6 Schematic architecture inside an FPGA. 18
1.7 Internal architecture of Cisco Flow Processor. 21
1.8 High-level structure of a 4 × 3-sized memory array. 23
1.9 SRAM bit cell circuit. 23
1.10 DRAM bit cell circuit. 25
1.11 High-level structure of CAM. 26
1.12 TCAM-based IP address lookup subsystem. 29
1.13 DRAM system overview. 30
1.14 Architecture of DRAM chip. 30
1.15 DRAM accessing without and with bank interleaving. 31
1.16 Basic mechanism of cache memory systems. 33
1.17 Architecture of LLC that consist of LLC slice connected via mesh inter-core bus. Inter-core bus is shown in green lines. . . . 34
1.18 Example of address mapping to LLC slices. 35
1.19 Structure of hybrid memory cube. 37
1.20 Structure of high bandwidth memory. 37
1.21 Chapter overview of this thesis. 45

3.1 Overview of proposed memory system architecture. 59
3.2 Main memory system of the proposed architecture. 60
3.3 State transition for each server. 64
3.4 State transition for each server in $N = 2$. 65
3.5 State transitions incoming to and outgoing from state (i, j, p). . 69
3.6 State transition diagram of IPP. 72
3.7 State transitions incoming to and outgoing from states $(i, j, p, 0)$, $(i, j, p, 1)$, and $(i, j, p, 2)$. 74
3.8 Blocking probability depending on ρ^{R} with different μ_2. 79
3.9 Blocking probability depending on μ_2 with $\rho^{\mathrm{R}} = 1.2$. 80
3.10 Throughput depending on ρ^{R} with different μ_2. 81
3.11 Effective waiting time depending on ρ^{R} with different μ_2. 81
3.12 Effective waiting time depending on μ_2 with $\rho^{\mathrm{R}} = 1.2$. 82
3.13 Blocking probability depending on l and h with $\rho^{\mathrm{B}} = 1.2$ and $\mu_2 = 0.7$. 83
3.14 Blocking probability depending on ρ^{B} and h with $l = 1.0$ and $\mu_2 = 0.7$. 84
3.15 Throughput dependency on l and h with $\rho^{\mathrm{B}} = 1.2$ and $\mu_2 = 0.7$. 84
3.16 Effective waiting time dependency on l and h with $\rho^{\mathrm{B}} = 1.2$ and $\mu_2 = 0.7$. 85

4.1 Overview of proposed architecture. 91
4.2 System model of proposed architecture. 93
4.3 System model of reference architecture with LLC. 95
4.4 System model of reference architecture without any cache. . . . 95
4.5 Blocking probability depending on traffic load for different architectures. 99
4.6 Blocking probability depending on memory capacity of L1 cache for different architectures. 100
4.7 Blocking probability depending on number of entries in each cache line for different architectures. 100
4.8 Average effective waiting time depending on number of entries in each cache line for different architectures. 101

4.9 Throughput depending on number of entries in each cache line for different architectures. 101

5.1 Overview of proposed architecture. 105

5.2 System model of proposed architecture, in which all queues, CPU cores, and LLC slices are assigned. 107

5.3 System model of proposed architecture, in which gray queues, CPU cores, and LLC slices are not assigned. 108

5.4 Example of system model of proposed architecture where $C_{\text{sys}} = 6$, $L_{\text{sys}} = 6$, $C = 5$, $L = 3$. There are ten packets P1, P2, $\cdots$, P10. 109

5.5 System model of reference architecture with LLC. 111

5.6 System model of reference architecture without LLC. 111

5.7 Example state of reference architecture with LLC, where $C_{\text{sys}} = 6$, $C = 5$. There are ten packets P1, P2, $\cdots$, P10. 112

5.8 Blocking probability depending on number of entries in each cache line. 116

5.9 Average effective waiting time depending on number of entries in each cache line. 117

5.10 Throughput depending on number of entries in each cache line. . 117

5.11 Blocking probability depending on number of assigned LLC slices. 119

5.12 Average effective waiting time depending on number of assigned LLC slices. 119

5.13 Throughput depending on number of assigned LLC slices. 120

5.14 Blocking probability depending on number of assigned CPU cores $B/b = 32$. 120

5.15 Average effective waiting time depending on number of assigned CPU cores $B/b = 32$. 121

5.16 Throughput depending on number of assigned CPU cores $B/b = 32$. 121

5.17 Blocking probability depending on total number of entries. . . . 123

5.18 Average effective waiting time depending on total number of entries. 123

5.19 Throughput depending on total number of entries. 124

5.20 Hit rate of each cache level depending on total number of entries. 124

6.1 Proposed architecture. 129
6.2 Example situation of proposed architecture. 131
6.3 Overview of system model. 132
6.4 System model of 3D-stacked DRAM. 133
6.5 Memory network for evaluation of dependency on number of packet processing tasks and traffic load. 137
6.6 Reference architecture for evaluation of dependency on number of packet processing tasks and traffic load. 138
6.7 Blocking probability depending on number of packet processing tasks and traffic load. 139
6.8 Throughput depending on number of packet processing tasks and traffic load. 140
6.9 Average effective waiting time depending on number of packet processing tasks and traffic load. 140
6.10 Memory network for evaluation of dependency on number of vertically cascaded 3D-stacked DRAMs for task 2 and traffic load.141
6.11 Blocking probability depending on number of vertically cascaded 3D-stacked DRAMs and traffic load. 142
6.12 Throughput depending on number of vertically cascaded 3D-stacked DRAMs and traffic load. 142
6.13 Average effective waiting time depending on number of vertically cascaded 3D-stacked DRAMs and traffic load. 143

List of Tables

3.1 State transitions incoming to and outgoing from state (i, j, p). . . 70

3.2 State transitions incoming to and outgoing from state $(i, j, p, 0)$ in IPP. 73

3.3 State transitions incoming to and outgoing from state $(i, j, p, 1)$ in IPP. 75

3.4 State transitions incoming to and outgoing from state $(i, j, p, 2)$ in IPP. 75

3.5 Example of packet processing performance. 86

Notations

Notation	Description
N	Number of banks in memory channel of 3D-stacked memory
S	Number of memory channels of 3D-stacked memory
K	Maximum number of memory requests in memory system
λ	Average arrival rate at memory system in Poisson arrival process
μ_w	Average service rate of each channel-bank set in 3D-stacked memory with $w \in [0, N]$-degree bank interleaving
μ_{L1}	Average service rate of on-chip L1 cache memory
μ_{L2}	Average service rate of on-chip L2 cache memory
μ_{LLC}	Average service rate of on-chip shared LLC and LLC slice
M_{L1}	Capacity of on-chip L1 cache memory
M_{L2}	Capacity of on-chip L2 cache memory
M_{LLC}	Capacity of on-chip shared LLC
M_{Slice}	Capacity of on-chip LLC slice
M_{3D}	Capacity of off-chip 3D-stacked memory
N_{ent}	Total number of database entries stored in 3D-stacked memory
C_{sys}	Total number of CPU cores in memory system
B	Data size of cache line
b	Data size of database entry
ρ	Traffic load of memory system
R	Set of memory requests issued by CPU cores during certain period of time
R_{b}	Set of blocked memory requests during certain period of time
r	Accepted memory requests during certain period of time

Notation	Description
t_r	Waiting time of each accepted memory request until it is processed by CPU core
P_b	Blocking probability of memory system
W_e	Average effective waiting time of memory system
λ_e	Throughput of memory system
L	Number of vertically cascaded 3D-stacked memories in memory network
M	Number of horizontally cascaded 3D-stacked memories in memory network
ω_i	Required number of memory accesses to process a packet for $i \in [1, M]$th packet processing tasks in memory network
T	Set of packets that enter memory network during certain period of time
T_b	Set of blocked packets during certain period of time
t	Accepted packet during certain period of time
τ_t	Waiting time of each accepted packet until it is processed in memory network and returned to FPGA
τ_tr	One-way data transfer latency between FPGA and 3D-stacked memory

Abbreviations

Abbreviation	Description
CAPEX	capital expenditure
OPEX	operating expenditure
COTS	commercial off-the-shelf
VNF	virtual network function
SLA	service level agreement
CPU	central processing unit
DRAM	dynamic random access memory
IP	Internet protocol
LLC	last-level-cache
SDN	software defined networking
NFV	network functions virtualization
ASIC	application specific integrated circuit
PLD	programmable logic device
NP	network processor
SRAM	static random access memory
CAM	content addressable memory
CRC	cyclic redundancy check
IPv4	Internet protocol version 4
MIB	management information base
UDP	user datagram protocol
SNMP	simple network management protocol
TOS	type of service
DPI	deep packet inspection
URL	uniform resource locator

Abbreviation	Description
ACL	access control list
TCAM	ternary content addressable memory
TTL	time-to-live
IPSec	Internet Protocol security
SSL/TLS	secure socket layer/transport layer security
VPN	virtual private network
HTTPS	hypertext transfer protocol secure
IoT	Internet of Things
5G	fifth generation
MEC	mobile edge computing
C-plane	control plane
U-plane	user plane
ONF	Open networking foundation
ONOS	Open networking operating system
NETCONF	network configuration protocol
YANG	yet another next generation
P4	programming protocol-independent packet processors
SRv6	segment routing IP version 6
ETSI	European telecommunications standards institute
ISG	Industry specification group
NFVI	NFV infrastructure
VM	virtual machine
KVM	kernel-based virtual machine
OSS/BSS	operation support system/business support system
MANO	management and orchestration
VNFM	VNF manager
VIM	virtualized infrastructure manager
HPC	high-performance computing
OS	operating system
RIB	routing information base
FIB	forwarding information base
USB	universal serial bus

Abbreviation	Description
SATA	serial advanced technology attachment
PCIe	peripheral component interconnect express
GPU	graphic processing unit
FPGA	field programmable gate array
UPI	Intel ultra path interconnect
ISA	instruction set architecture
NIC	network interface card
LSI	large-scale integrated circuit
LB	logic block
LUT	lookup table
JTAG	joint test action group
HDL	hardware description language
RTL	register transfer level
PPE	packet processor engine
RISC	reduced instruction set computer
SIMD	single instruction, multiple data
RAM	random access memory
NVRAM	non-volatile random access memory
STT-MRAM	spin-torque transfer random access memory
PCM	phase change memory
ReRAM	resistive random access memory
FeRAM	ferroelectric random access memory
NRAM	nanotube random access memory
BCAM	binary content addressable memory
DMA	direct memory access
DIMM	dual inline memory module
L1	level 1
L2	level 2
LRU	least-recently-used
CAT	cache allocation technology
TSV	through silicon via
DDRx	double data rate x

Abbreviation	Description
HMC	hybrid memory cube
HBM	high bandwidth memory
PIM	processing-in-memory
DPDK	Data plane development kit
SR-IOV	Single root I/O virtualization
TCP	transmission control protocol
FCFS	first come first served
IPP	Interrupted Poisson Process

Chapter 1

Introduction

1.1 Packet processing

1.1.1 Packet processing types

A network consists of various network equipment, each of which corresponds to some of the network functions such as routers, switches, gateways, firewalls, and load balancers. Each network function comprises several types of packet processing tasks between the arrival and departure of the packet. Examples of packet processing tasks include framing, parsing, classification, searching, forwarding, modification, queueing, and traffic management [1]. Note that not all of these packet processing tasks are always required for every network functions. The required packet processing tasks are decided according to the network functions that the equipment has. Each packet processing task has to understand the information of the packet, update the packet content, or lookup the database in the memory. Therefore, packet processing tasks issue memory accesses to read/write the packet content and the data stored in the memory.

The rate of performance improvement in processors has been higher than that in memories [2–8], which increases the importance of memory performance in packet processing. The more memory requests are processed in a certain time, the more packets are processed. Therefore, as we will see in Section 1.3, dedicated system architectures are adopted in dedicated network equipment, where several function dedicated memories and parallel memories are assigned for some memory-intensive tasks such as classification, searching, and queueing.

In the following subsections, this thesis presents brief descriptions of each packet processing task. This thesis uses the word "packet" in a broad sense, which means a generic term of independent data unit in a network.

1.1.2 Framing

In the framing process, the received or transmitted packets undergo several processes. For instance, packets receive the following processes: error detection to make sure that the bits in the received packets have no error, error corrections if there are some bit errors when the packets have redundant information for corrections, and fragmentation/assembly to transmit/receive the data larger than the maximum per-packet data size, which is defined in the protocol of the packets. Thus the framing process is usually located in the first or last step of packet processing.

For error detection, there are several algorithms, each of which is used for different purposes with considering its calculation complexity and reliability. For example, Ethernet uses cyclic redundancy check (CRC) algorithm [9] in the frame check sequence field of the packet. The CRC is based on the remainder calculation of a modulo-2 division of a polynomial with coefficients of 0 and 1 by an agreed-upon generator polynomial. The CRC assumes that the packet's bit string represents a polynominal with coefficients of 0 and 1. The remainder is sent with the packet and compared with the recalculated remainder using the same agreed-upon generator polynomial at the other end. If the remainder is 0, there is no error in the received packet; otherwise it is wrong. Typically, n-bit CRC can detect any single error burst not longer than n bits. Ethernet uses 32-bit CRC, usually called CRC-32, which is defined in IEEE 802.3 [10]. On the other hand, IP uses simple checksum, where the sum of all the specific bits of a packet is calculated [11]. The checksum is sent with the packet and compared with the recalculated one at the other end. If the two are equal, the received packet has no error; otherwise it has errors.

When an original packet is fragmented into multiple packets, additional information is added to the headers of each fragmented packet so that each fragmented packet can travel the network independently. In the case of IP, the additional information corresponds to the FLAGS and FRAGMENT OFFSET fields, which show if the packet is fragmented or not and the position of the packet in the fragmented packets, respectively. When the fragmented packets are received at the other end, the receiver assembles them into the original packet using the additional information in the packet headers of each fragmented packet.

1.1.3 Parsing and classification

In parsing and classification processes, packets are analyzed and classified according to the contents of the packets. Usually, parsing cooperates with classification.

Packet parsing examines the packet to understand the packet structure and identify each field of the packet. For example, the destination IP address of an Internet Protocol version 4 (IPv4) packet has to be identified in the parsing process, which will be used to lookup the routing table and forward the packet in searching and forwarding processes. A more complex parsing example is to detect several management information base (MIB) variables in some of the IPv4 packets which contain user datagram protocol (UDP) packets running simple network management protocol (SNMP) in order to classify these packets for the network operation.

Packet classification categorizes the packets into several "flows" according to some of the fields and/or the content in each packet. In each flow, the classified packets go through similar packet processing. For instance, the type of service (TOS) field of an IP packet is used to prioritize the incoming packets. In addition, multiple fields of an incoming packet are used for the packet classification. A set of five different values which consist of the source IP address, the source port number, the destination IP address, the destination port number, and the protocol, usually called 5-tuple, is commonly used to filter the incoming packets. Moreover, deep packet inspection (DPI) function is a complicated example of packet classification, where the packet content is inspected to detect any specific uniform resource locator (URL) or keywords for packet filtering.

1.1.4 Searching and forwarding

Searching is one of the most important operations in packet processing. Usually, searching operation is also included in packet classification, forwarding, and queueing to decide the appropriate action for the packet, which means that searching is an atomic operation rather than an independent packet processing task. For packet classification to filter the incoming packet, an access control list (ACL) that contains some packet processing rules must be searched. The ACL is a database usually formed as a table, whose searching key is the detected packet information at the packet parsing process. For IP packet forwarding, IP address lookup is required, where the key is the destination IP address of the packet.

Methods for the searching operation are categorized into software-based methods and hardware-based methods. In software-based methods, they are distinguished based on their data structures of the database and algorithms. Examples of commonly used structures are, list structures such as linked lists and skip lists, table structures such as hash tables and associative arrays, tree structures such as binary search tree, and tries [12] such as Patricia tries and multi-bit tries.

In hardware-based methods, dedicated hardware devices, usually called search engines, such as content addressable memory (CAM) and ternary CAM (TCAM) are used. In each method, the fundamental operations for the data handling are as follows: to search the data structure and get the result, to insert an additional entry to the database and maintain the data structure, to delete an entry from the database, to modify the content of the entry in the database. Each software/hardware-based method has different characteristics such as time complexity to perform an operation, size efficiency of the database, and scalability. In addition, power efficiency, device cost, and peripheral circuits around the search engines are also taken into account for hardware-based methods.

In particular, IP address lookup requires the longest prefix match (LPM), which has been one of the most complicated, common, and important operations in packet processing. In the LPM, the longer the match is, the better the routing that will be selected for the packet. This means that the packet processing system has to search the best-matched entry in the entire database rather than the just matched one. There have been several works and hardware/software implementation for the LPM in IP address lookup [13–17], which shows the difficulty and importance of the searching operation in packet processing.

1.1.5 Modification

Packet modification is the operation to edit the packets, which is sometimes a part of other packet processing tasks. The fundamental packet modification operations are as follows.

- Changing the contents of the packets such as some header fields and the payload to update the time-to-live (TTL) filed, IP addresses fields, the checksum of an IP packet, and so on.
- Deleting some packet content or headers for such as de-encapsulation of packets.
- Adding additional information to the packets for such as encapsulation.
- Removing the entire packet for such as dropping packets that do not pass the filtering based on an ACL and dropping bandwidth exceeded traffic.
- Copying the entire packet for such as multicasting and port mirroring.

1.1.6 Encryption

Encryption is used to maintain data privacy and data integrity and to authenticate the identities of the communicating parties. There are several protocol standards for secure communication, such as Internet Protocol security (IPSec) [18], secure socket layer/transport layer security (SSL/TLS) [19], and so on. IPSec is usually used in virtual private network (VPN) communications.

SSL/TLS is typically used in client-server applications, such as web communication in hypertext transfer protocol secure (HTTPS) and secure email services.

Packet processing in such protocols for secure communication is the transformation of information from unsecured information to coded information and vice versa by using some key and the transformation algorithm. To execute such transformation algorithm requires additional computing power, which makes packet processing that includes security protocol processing difficult to run at wire speed. Therefore, packet encryption is often implemented using a dedicated encryption engine in a processor or in a discrete chip.

1.1.7 Queueing and traffic management

Usually, queueing and traffic management are the last tasks in packet processing just before the packet transmission. In classification, searching, and modification, the parameters of the packet such as the destination port and the priority are determined according to the results of each processing. In queueing and traffic management process, packets are forwarded to the appropriate queues and scheduled for the transmission according to the parameters of the packet and other conditions of the receivers and the transmission lines, and so on. Also, queueing and traffic management process meters the packets and shapes the transmission pattern to an intended rate and burstiness.

1.2 Network virtualization

Internet traffic has been growing due to the increasing number of Internet users and connected devices. This trend is considered to be continue [20, 21]. New network services such as Internet of Things (IoT), fifth generation (5G) mobile networks, and mobile edge computing (MEC) increase the network traffic and diversify the traffic patterns and requirements of networks [22–29]. Therefore, network equipment should be more scalable to accommodate the increasing network traffic and be more adaptive to the rapid deployment of new network functions.

Network virtualization is the technological movement to realize more flexible and efficient network infrastructure by using software-defined network operation and inexpensive COTS hardware. Network virtualization roughly consists of software defined networking (SDN) approach and network functions virtualization (NFV) approach [30], each of which is explained in the subsequent subsections.

1.2.1 Software defined networking (SDN)

SDN is the separation of the control plane (C-plane) and the user plane (U-plane), also known as data plane, in network equipment. In conventional networks, a network operator has to directly configure each network equipment and physically change the network connection such as cabling. On the other hand, by separating C/U-plane in physical network equipment, a network operator can construct and manage logical networks by programming each network equipment via an SDN controller without changing the underlying physical network.

In addition, the C/U-plane separation enables the network operator to use the commodity hardware such as white-box switches that have merchant silicon devices instead of conventional purpose-built network equipment. The well-known examples of the merchant silicon devices are Broadcom's switch chips such as the Trident series [31], which has more than several Tbps switching capacity and some simple packet processing functions such as forwarding and encapsulation. A network operator can program these chips in the SDN-aware switches and routers via the SDN controller to design and construct desired virtual networks.

Protocols and languages are provided between an SDN controller and SDN-aware network equipment in order to program the network equipment in a flexible and efficient way without considering the specific vendors and models of each network equipment. The flexible and efficient programming scheme also allows the network operator to choose the network equipment from more various vendors and models than the conventional vertically integrated network equipment.

A well-known protocol example is OpenFlow [32], which is standardized in Open Networking Foundation (ONF). The corresponding SDN controllers are

ONF's Open Network Operating System (ONOS) [33]. Faucet [34] is another SDN controller for OpenFlow. There are several software switches that can be programmed using OpenFlow, such as Open vSwitch [35] and Lagopus [36], whereas there are some hardware switches and routers that are compatible with OpenFlow. In OpenFlow architecture, C-plane functions are centerlized in OpenFlow controller, which means that the same OpenFlow controller directly controls all the OpenFlow switches and routers in the same network.

Network configuration protocol (NETCONF) [37] is another well-known protocol for SDN. Yet another next generation (YANG) [38], a data modeling language, is usually used to write the configurations for the NETCONF-compatible switches and routers. In contrast to OpenFlow, the NETCONF-based SDN scheme provides programmability of network equipment, whereas C-plane functions still work in network equipment without moving them to the SDN controllers. The NETCONF-based SDN scheme allows each switch or router behaves independently according to the programmed configurations. To program network equipment, the vendors release plugins for NETCONF-compatible SDN controllers to obscure the particular parameters and attributes of each equipment so that the network operator can construct the network without being conscious of the vendors/models differences.

In addition to C/U-plane separation in network equipment, U-plane functions of the chips are becoming more open and programmable compared to conventional dedicated network equipment. Programming protocol-independent packet processors (P4) [39] is a high-level language that enables the users and operators of networks to design and program the packet processing functions to the chips without being conscious of the type of processors and the details of packet processing. In conventional network equipment and its chip, the implemented U-plane functions are determined by the chip designer, which sometimes prevents the network users and operators from using their desired functions due to its lack of programmability. Segment routing IP version 6 (SRv6) [40], a type of source routing protocols, is an example of emerging networking protocols, which is first implemented in a software router [41] and then implemented in the physical hardware routers. Thus SRv6 is suitable for hardware implementation using programming language such as P4. In fact, several projects implemented SRv6 functions to hardware devices using P4 language [42]. If U-plane functions are fully and easily programmable by network users and operators, both C/U-plane are programmable by using SDN technologies, which makes network equipment as open and flexible as the general-purpose computers.

1.2.2 Network functions virtualization (NFV)

NFV is the separation of software and hardware in computer networks, where the network functions are extracted as software, called virtual network functions, and the hardware of conventional purpose-built network equipment is replaced with inexpensive general-purpose computers. With this separation of software and hardware, the network operator can operate the network in more flexible and efficient way. For instance, the operator can migrate VNFs running on a computer to another computer when there is trouble in the original computer and dynamically increase/decrease the number of VNF instances according to the network service demand.

NFV is standardized at European Telecommunications Standards Institute (ETSI) NFV Industry Specification Group (ISG). NFV standardization has been conducted since its first proposal [30] in 2012. Figure 1.1 shows an NFV reference architecture [43], where conventional network infrastructure is broken down into several fundamental components. VNFs work on top of the NFV infrastructure (NFVI). NFVI consists of physical hardware resources, a virtualization layer, and virtual resources that comprise virtualized computers. The physical hardware resources include computing hardware such as CPUs and memories, storage hardware, and network hardware. The virtualized computers are usually instantiated as virtual machines (VMs) and containers. Also, the virtualized computers are mapped to the physical resources via the virtualized layer software such as kernel-based virtual machine (KVM) for VMs and Docker for containers. Operation support system/business support system (OSS/BSS) are the components that are used by telecom operators for order management, network inventory management, billing, customer relationship management. NFV management and orchestration (MANO) manages the lifecycle of all the components of NFVI and VNFs via several interfaces shown in Figure 1.1. NFV MANO consists of an orchestrator, VNF managers (VNFMs), and a virtualized infrastructure manager (VIM). Usually, VNF vendors release the corresponding VNFM for the VNF, whereas there are approaches to realize a generic VNFM [44, 45]. The VIM manages the resources of NFVI in cooperation with the VNFMs and the orchestrator. OpenStack [46] is a de facto standard of the VIM, which is widely used in commercial networks. NFV MANO is the main scope of the NFV standardization at ETSI NFV ISG, which means that other components such as NFVI and VNFs are not standardized in detail at ETSI NFV ISG.

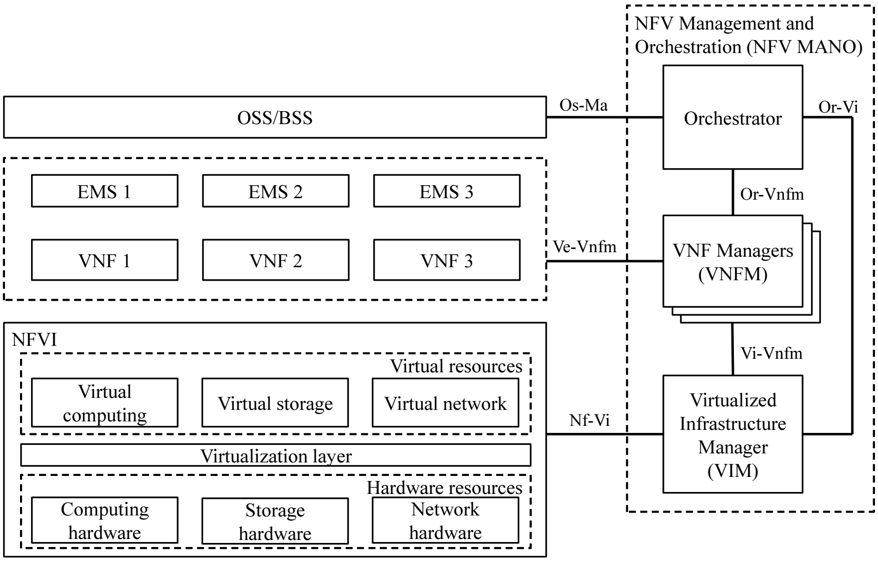


Figure 1.1: NFV reference architecture.

In order to benefit from NFV, performance of VNFs has to satisfy the requirements of network services even when the VNFs run on top of the virtualized infrastructure. When implementing a network function that was previously on top of the dedicated equipment as an application software of general-purpose computers, architecture differences between the dedicated hardware and the general-purpose one must be taken into account so that the VNF on top of general-purpose computers properly work with acceptable performance, which leads to the motivation of this thesis.

1.3 Computer architectures for packet processing

Not only general-purpose computers but also dedicated network equipment such as switches and routers are computers, whereas there are several architecture differences in terms of the processors, memories, networking interfaces, interconnection between the devices, and the processing parallelism. This section briefly explains the computer architectures of conventional dedicated equipment and general-purpose computers for packet processing.

Note that there are two basic design philosophies for packet processing; store-and-forward and cut-through. In the store-and-forward method, the received packet is first stored in the memory temporarily and goes through several packet processing tasks. After the decision is made for the packet, the packet is transmitted and removed from the memory. Since the store-and-forward method buffers the entire packet, the packet processing system can perform complex and detailed packet processing. In particular, DPI is an example of complex and detailed packet processing and requires parsing, classification, and queueing. On the other hand, packet buffering in the store-and-forward method introduces additional packet processing latency.

In the cut-through method, packet processing begins as soon as the packet arrival by examining some specific bit patterns in the packet in real-time. While the cut-through method minimizes the buffering latency compared to the store-and-forward method, it only allows simple packet processing based on the limited field in the packet, where complex packet processing such as DPI and CRC cannot be implemented. Usually the store-and-forward method is applied when multiple packet processing functions are needed for network services. On the other hand, the cut-through method is selected when lower buffering latency is preferred rather than rich packet processing functions in networks for high-performance computing (HPC) [47–49].

In this thesis, computer architectures for packet processing are supposed to be based on the store-and-forward method.

1.3.1 Dedicated architecture

In order to cope with a large traffic of various network services, conventional network equipment has dedicated computer architecture, where purpose-built processors, memories, and interconnection among chips are utilized. Figure 1.2 shows a high-level computer architecture of conventional network equipment, where incoming packets are processed in parallel at the packet processing processor with the assistance of the several peripheral devices connected to the packet processing processor [50].

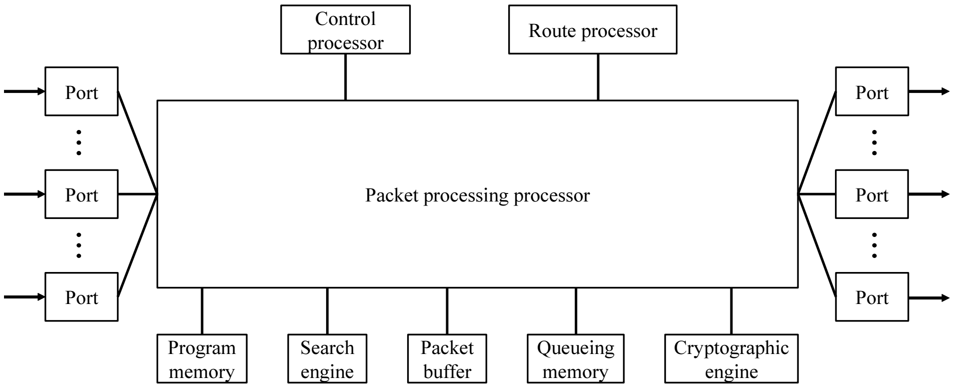


Figure 1.2: High-level computer architecture of conventional network equipment.

While the details of the packet processing processor are explained in Section 1.4, it conducts some of the packet processing tasks such as framing, parsing, classification, searching, modification, encryption, queueing, and traffic management as described in Section 1.1. The peripheral devices connected to the packet processing processor are categorized into two types; the devices for C-plane processing and the devices for U-plane processing. For C-plane processing, there are usually control processors and dedicated route processors. The control processor is usually a general-purpose CPU and is used to run the operating system (OS) of the entire equipment, which enables the network operators and developers to control and program the equipment. While the route processor is usually a general-purpose CPU as well, whose purpose is different from the control processor. The route processor manages and updates the routing information base (RIB) and forwarding information base (FIB) by communicating other network equipment. For U-plane processing, there are several peripheral devices for packet processing. The program memory contains the program or microcode of packet processing software executed by the packet processing processor. The search engine is an accelerator of the classification and searching process. Usually, a TCAM is used because a TCAM can return the lookup result always in a certain number of clock cycles. The mechanism of a TCAM is explained in Section 1.6. A packet buffer and a queueing memory are used in the queueing and traffic management process for traffic shaping and QoS control. Since the queueing and traffic management process requires fast memory accesses and sufficient memory capacity for queueing, static random access memory (SRAM) is usually used. A cryptographic engine is an accelerator device of cryptography, in which complex calculation is conducted by using a dedicated circuitry.

Figure 1.3 shows a high-level system architecture of modular network equipment, which is usually used in aggregation networks and core networks of telecom networks or service provider networks, in which a large amount of traffic is assembled. In order to augment packet processing performance of the entire system, such modular network equipment comprises multiple line cards, each of which corresponds to the basic architecture shown in Figure 1.2.

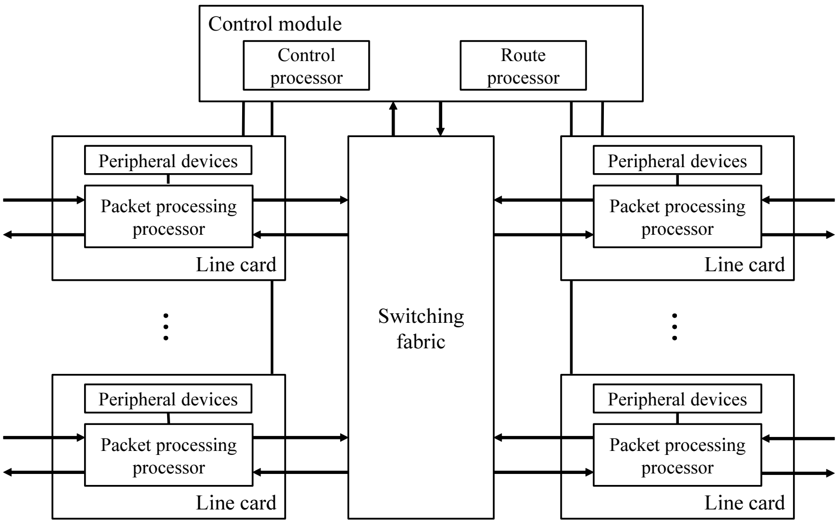


Figure 1.3: High-level system architecture of a modular network equipment.

C-plane components such as the control processors and route processors are not usually included in each line card and separate in the control module. All the line cards are connected to the switching fabric or backplane of the rack so that they can communicate with one another. The control module is also connected to the switching fabric so that some packets can be sent to and from the control module for routing information updates and other signaling. In addition, the control module is connected to every line card via a dedicated interface that is different from the switching fabric.

This system architecture allows the network operator to scale packet processing performance to meet network service demands by increasing or decreasing the number of line cards. In addition, some of the modules and line cards may be standby, whereas the others are active to increase the system availability.

1.3.2 General-purpose architecture

Apart from dedicated network equipment, general-purpose computers are used to execute various application software, which is not limited to packet processing. Figure 1.4 shows a schematic computer architecture of general-purpose computers based on Intel x86 CPU. There are CPUs, DRAMs as CPU's main memories, a chipset that integrates connection between the CPU and the peripheral devices such as universal serial bus (USB) for various devices and serial advanced technology attachment (SATA) for storage devices, and peripheral component interconnect express (PCIe) for other accelerator devices such as graphic processing units (GPUs) or field programmable gate arrays (FPGAs).

The CPU has multiple CPU cores, on-chip cache memories, a DRAM controller, PCIe connections, and the inter-CPU link such as Intel ultra path interconnect (UPI). Modern servers based on x86 CPUs can install multiple CPUs, usually up to eight CPUs depending on the specifications of CPU, in the same computer hardware, in which each CPU is connected via the inter-CPU link. The details of CPU, DRAM, memory systems, and FPGA are explained in Sections 1.4.4, 1.5.2, 1.7, 1.4.2, respectively.

Unlike dedicated network equipment, general-purpose computer architecture is not designed solely for packet processing. Instead, hardware and software of general-purpose computers are separated clearly, which means that

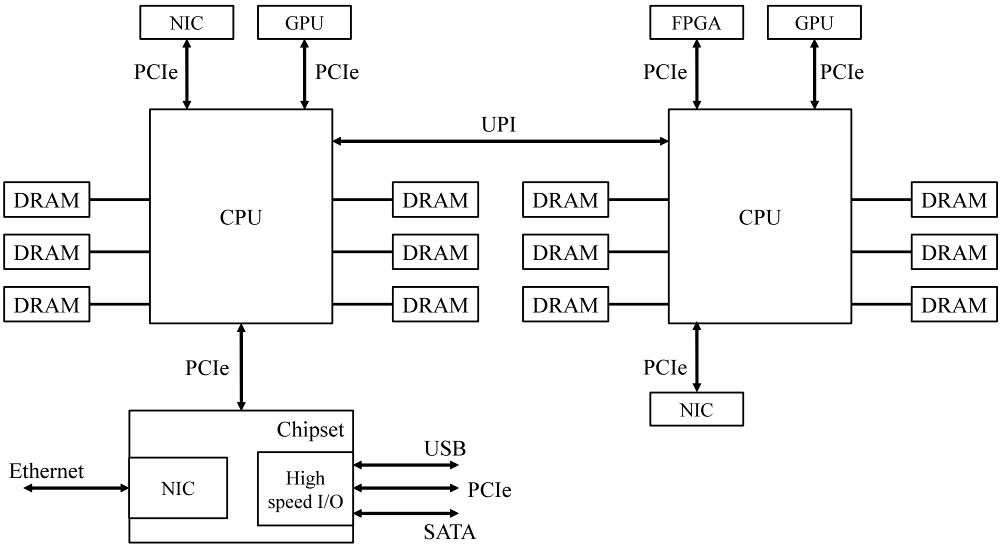


Figure 1.4: High-level architecture of general-purpose computers based on x86 CPU.

various application software can be executed on different computers by using the compiler which corresponds to a specific instruction set architecture (ISA) of the CPU. The CPU is the center of the general-purpose computer architecture, in which most of the packet processing is usually done, which means that C-plane processing and U-plane processing are not separated in the hardware architecture. Therefore, it is common to assign separate CPU cores to U-plane processing and C-plane processing, or U-plane and C-plane processing are performed on different computers. In addition, packet processing flow is not determined in general-purpose computer hardware, which means that every packet treatment such as the reception of incoming packets, packet parsing, classification, lookup, modification, queueing, shaping, and packet transmission is described in software including the OS of the computer.

There are several approaches to increase the performance of each packet treatment, some of which are explained in Section 2.1. Moreover, although some of the packet processing tasks such as checksum calculations are offloaded to the dedicated hardware in the network interface card (NIC), there are no dedicated engines or circuitry for specific processing such as traffic management and searching in general-purpose computers. Due to the lack of purpose-built hardware, packet processing performance on general-purpose computers suffers a significant handicap compared to the dedicated network equipment.

1.4 Processor types in computer systems

A processor is one of the essential parts of every computer architecture. This section explains the basic knowledge of the processor devices. Note that the word “processor” is used in the broad meaning, which includes the hardware devices that execute a software program and the integrated circuit devices in packet processing systems.

1.4.1 Application specific integrated circuit (ASIC)

ASIC is the purpose-built integrated circuit device which contains the required circuitry for multiple functions. The desired functions are directly implemented as hardware circuitry instead of software programs on top of processors, which makes ASIC high-performance in terms of throughput, latency, and power efficiency. Therefore, dedicated network equipment in telecom networks and large service provider networks depends on the ASIC for high-performance packet processing to satisfy the SLA.

On the other hand, since circuitry in ASIC can never be modified once it is designed and manufactured, ASIC usually requires very careful, long-term, and costly development. Therefore, if network functions are implemented in the ASIC of network equipment, even a small function addition requires such long time-to-market of ASIC, which prevents the network operator from using the latest network protocol and rapidly adopting the change of traffic patterns and requirements due to the emerging network services.

Network virtualization aims to replace the ASIC and its associated dedicated system architecture with general-purpose computers and software-based network functions as many as possible for CAPEX reduction. Additionally, network virtualization also lowers the ASIC usage in network equipment so that ASIC is only used in simple layer two or layer three switches, controlled in a software-defined way, which process large amount of traffic using limited packet processing functions. Figure 1.5 shows a schematic architecture of such simple switch ASIC device [51], in which each ingress and egress pipeline executes fixed packet processing such as parsing, tunnel termination, switching, and modification [52].

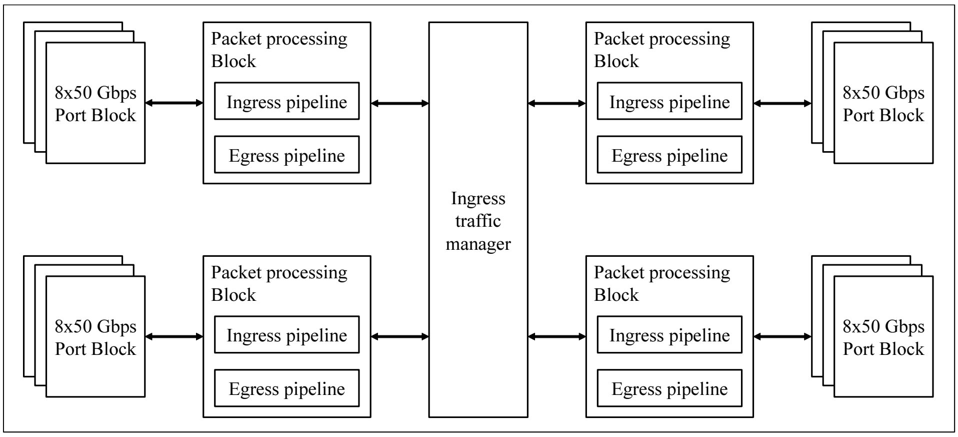


Figure 1.5: High-level architecture of switch ASIC.

1.4.2 Programmable logic device (PLD)

PLD is a general term for large-scale integrated circuit (LSI) device, in which a designer can deploy desired logical circuits for multiple times by programming. While PLDs are programmable, there may be some circuits that cannot be implemented in particular PLDs depending on the size of the circuits, the operating frequency of the PLDs, and delay requirements of the circuit design, which requires higher-performance PLDs with a larger number of logic elements and higher operating frequency. This thesis mainly focuses on FPGAs among PLDs, which have a large circuit that can be used for packet processing.

Traditionally, FPGAs have been used as auxiliary devices, such as prototyping ASICs and circuits to connect between ASICs and other peripheral devices, in order to avoid the long-term and costly development of ASICs. The increased integration density of semiconductor devices has increased performance and circuit capacity of FPGAs, which makes FPGAs to be used for the acceleration of general-purpose processors, such as latency reduction and power efficiency improvement, instead of using ASICs.

An FPGA is composed of arrays of logic blocks (LBs), IO blocks, and the wiring between them including switches. Figure 1.6 shows a schematic architecture inside an FPGA. The logic block consists of a lookup table (LUT) that contains a truth table corresponding to a combinational circuit and a flip-flop circuit as a register that stores the output of the combinational circuit. The LUT usually has four inputs and one output, and the output is connected to the register input. The inputs to the LUT specify the address of the truth table, and the corresponding value of the specified address represents the behavior of the desired combinational circuit such as AND, OR, and more complicated operations.

To program an FPGA, based on the programming language describing the desired logic circuits, compilers or design tools determine the optimal placement of logic operations to logic blocks and the routing between the logic blocks with considering the actual FPGA pin assignment including hard-coded dedicated high-speed interface circuits, delay requirements directed by the designer, and the circuit size compared to the circuit capacity of the FPGA. After the placement and routing, the LUT values of each assigned logic block is configured via dedicated interfaces such as joint test action group (JTAG) or loaded from memory devices connected to the FPGA.

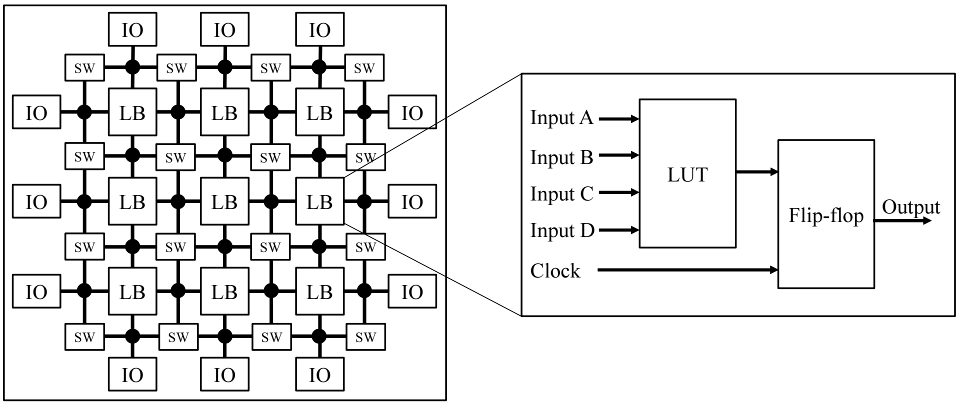


Figure 1.6: Schematic architecture inside an FPGA.

The hardware description language (HDL) is used to describe the logic circuits to be implemented in FPGAs or ASICs. There are several different levels of abstraction in HDLs, which range from the level of system architecture and algorithms to the level of flip-flops and logic gates. In FPGAs and ASICs development, the register transfer level (RTL) is usually used, which describes registers, logic operators, and the wiring between them. Therefore, when describing the logic circuits in RTL, the proper usage of combinational circuits and sequential circuits, clock synchronization are must be explicitly taken into account. Also, it must be carefully verified that the designed circuits can operate at the desired timing and speed on the target FPGA before implementation. Typically, FPGA development tools have target-specific timing simulators and verification features. Along with the programming of the logic circuits, it is also necessary to define the pin assignment, which maps the logic circuits to be implemented in the FPGA to the actual FPGA pins, based on the schematics of the circuit board on which the FPGA is mounted.

Although hardware development using HDLs is generally considered to be difficult because various hardware-related knowledge is required, easy-to-implement methods such as programming logic circuits in high-level languages are also developed. There are several tools from multiple FPGA vendors that compile program written in high-level languages, such as C and C++, into HDL. The P4 language explained in Section 1.2.1 is also an easy-to-implement language specialized in the description of packet processing applications. It converts the high-level description of packet processing into a programming language suitable for various chips including FPGAs. Therefore, due to these emerging methods to program FPGAs in high-level languages, FPGAs have attracted as much attention as general-purpose processors in network virtualization. There are several products such that integrate FPGAs into CPUs, NICs on which FPGAs are mounted, and PCIe extension cards that have FPGAs in general-purpose computers.

1.4.3 Network processor (NP)

NPs are the processors that are specialized in packet processing and are controlled by software. However, unlike general-purpose processors, NPs are designed to achieve packet processing at wire rate. Traditionally, packet processing at wire rate is achieved by using ASICs associated with costly development and long time-to-market. Therefore, NPs have gradually replaced the ASICs in packet processing.

Compared to general-purpose processors, NPs have different processor architecture and programming model. The architectural features of NPs include the parallel and pipelined packet processing, dedicated circuits for such as traffic management, and interfaces to external hardware engines such as TCAMs for searching, SRAMs for queueing. The programming models of NPs depend on the NP vendors and the processor architectures in terms of heterogeneity and complexity of each NP. In order to satisfy the strict performance requirements such as packet processing at wire rate, programming models of NPs are usually based on real-time programming, which includes event-driven processing according to the incoming packets, parallel and pipelined packet processing, and non-preemptive scheduling. In particular, a programmer of NPs may be required to program with an explicit awareness of the the parallel and pipelined architecture, which may lead to programming in low-level languages or vendor-specific microcode. Therefore, programming NPs is considered to be more difficult than programming general-purpose processors. On the other hand, some NP vendors release their development tools that obscure the processor architecture so that the packet processing can be programmed in high-level languages such as C-language.

Cisco Flow Processor [50] is an example of NP, which is used in the aggregation routers of telecom networks and routers for enterprise and service providers, whose system architecture is shown in Figure 1.2. Figure 1.7 shows the internal architecture of Cisco Flow Processor [50]. In the NP, there are 40 packet processor engines (PPEs), each of which has four threads and can be programmed in C-language. Each PPE thread executes the necessary packet processing program according to each incoming packet. Each PPE and its threads can work in parallel and are not a fixed pipeline. The PPE is a reduced instruction set computer (RISC)-based 32-bit core, which does not include the dedicated queueing and scheduling functions. Therefore, there is a dedicated traffic manager with a dedicated queueing engine. The NP also has several high-speed interfaces, to which TCAMs for fast searching and classification, DRAMs for packet buffering, SRAMs for queueing and scheduling, and encryption engines are connected.

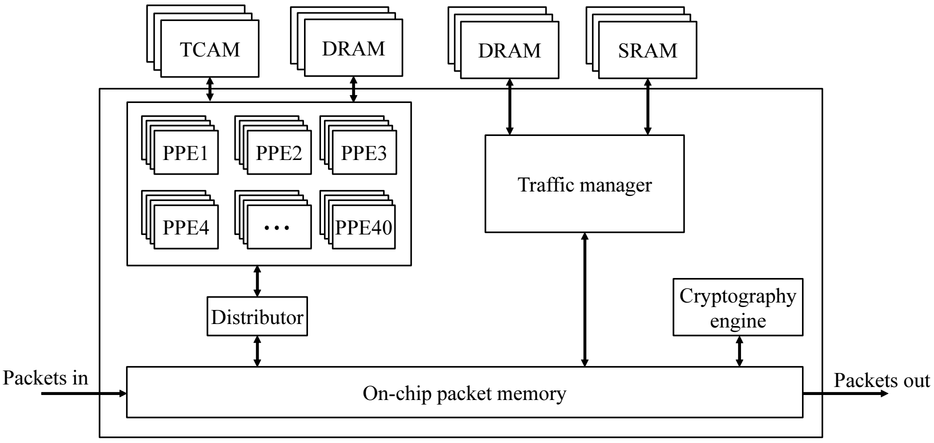


Figure 1.7: Internal architecture of Cisco Flow Processor.

1.4.4 Central processing unit (CPU)

CPU is the central processor in general-purpose computers and executes software programs. Intel x86 architecture-based CPUs are widely used in personal computers and general-purpose servers, whereas ARM architecture-based CPUs are also widely used in mobile devices and embedded systems. This thesis mainly focuses on x86 architecture-based CPUs, which are used in general-purpose servers in network virtualization.

CPUs are not built to specialize in a particular process but have basic arithmetic instructions required for a wide range of programs. While pipelining is usually utilized in CPUs, their programming models do not require explicit awareness of pipelining. Modern CPUs also have parallel processing features such as single instruction, multiple data (SIMD) execution, multiple CPU cores with multiple threads. These features are usually used by optimization functions of compilers and some of the OS functions rather than by explicitly describing in programming languages.

In x86 architecture-based CPUs, there are several to dozens of CPU cores, each of which is capable of running two threads. The CPU has a main memory system that consists of DRAM modules, on-chip several levels of cache memories, some of which are dedicated to each CPU core and the other is shared among all the CPU cores. The details of the main memory system and on-chip cache memories are explained in Sections 1.7.1 and 1.7.2, respectively.

Since CPUs are not designed for specific processing, CPUs in general-purpose servers perform packet processing in cooperation with peripheral devices connected via PCIe interfaces. As described in Section 1.3.2, a CPU in general-purpose computers is connected to the NICs, FPGAs, and GPUs. These CPUs, NICs, and the memory system are the key devices in general-purpose computer hardware to perform packet processing of VNFs. In particular, packets are received from the NIC and stored in the memory, tables in the memory are searched, and the packets in the memory are modified. FPGAs and GPUs may be used for acceleration of some specific packet processing tasks, such as parsing, searching, and so on [53, 54].

1.5 Memory types in computer systems

Memory devices in computer systems have an array structure to accommodate large data effectively. A memory array usually has a pair of $\boldsymbol{P}$-bit address input and $\boldsymbol{Q}$-bit data port, which consists of $2^{\boldsymbol{P}} \times \boldsymbol{Q}$ bit cells. Each bit cell stores one-bit data and is connected to a word line and a bit line. Figure 1.8 shows the high-level structure of a 4×3-sized memory array, in which there are four word lines corresponding to two-bit address input and three bit lines corresponding to three-bit word width. When a data in a specific word is read out from the memory array, the corresponding word line is pulled up, then the bit cells in the corresponding row drive each bit line. The data can be retrieved by reading the state of each bit line. When a data is written to the memory array, the bit lines are driven according to the data to be written, and then a specific word line is pulled up. The data in the bit lines are written to the bit cells in the row whose word line is in a high state.

1.5.1 Static random access memory (SRAM)

An SRAM bit cell in the SRAM array consists of a pair of CMOS-based inverters connected to a word line, a bit line, and an inverse of the bit line. Figure 1.9 shows a schematic SRAM bit cell in (a) logic circuit and (b) transistor-level circuit, where WL, BL, $\overline{\text{BL}}$ represent a word line, a bit line, and an inverse of the bit line, respectively [55].

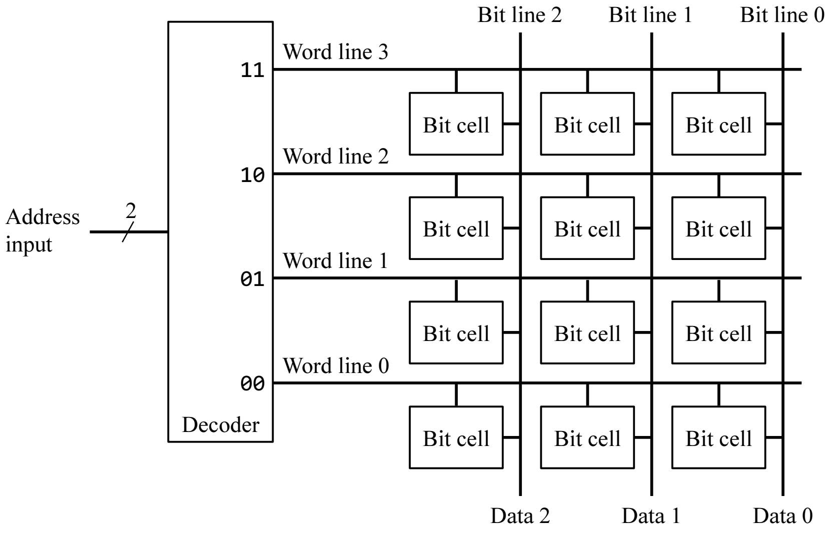


Figure 1.8: High-level structure of a 4 × 3-sized memory array.

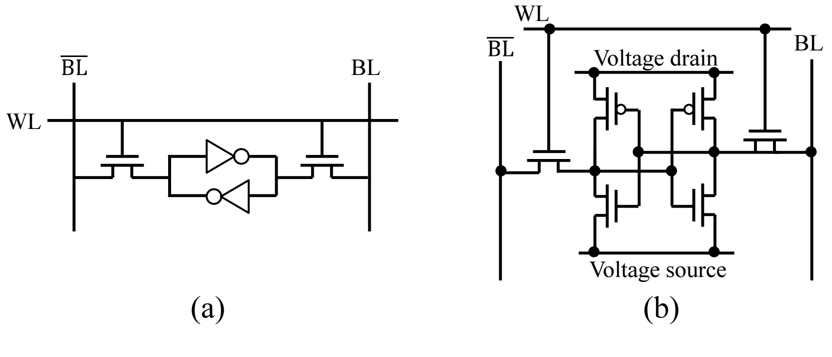


Figure 1.9: SRAM bit cell circuit.

The bit date is stored in the pair of CMOS-based inverters. When the word line is activated, the pair of inverters is connected with the bit line and its inverse, which allows the data to be read out or written. While the inverse of the bit line is not necessary in principle, it is usually used in addition to the bit line in order to improve the signal to noise ratio. Additionally, by measuring the electrical potential difference between the BL and $\overline{\mathrm{BL}}$, the data can be read out faster and more effectively than sensing the voltage of BL alone.

Accessing SRAM is faster than accessing DRAM based on a capacitor as explained in Section 1.5.2. On the other hand, as shown in Figure 1.9 (b), an SRAM bit cell requires six transistors to store one bit, which degrades the area efficiency of the SRAM array. Based on these characteristics, SRAMs are usually used for the cache memories in the processors and LUTs of FPGAs, where memory speed is preferred to memory capacity. Note that there is another SRAM bit cell circuit, in which only four transistors are used. However, four transistor SRAM is not used in memory devices of computer systems such as cache memories due to the high standby leakage current that increases the power consumption. While other implementation methods to reduce the standby leakage current of four-transistor SRAMs have been considered [56], in any case, an SRAM bit cell requires a relatively large number of transistors and corresponding area per bit.

1.5.2 Dynamic random access memory (DRAM)

A DRAM bit cell consists of a capacitor and a transistor, in which the capacitor is connected to a bit line whose connection is switched by the transistor. The capacitor stores one-bit data depending on whether the capacitor is charged or not. Figure 1.10 shows a schematic circuit of a DRAM bit cell, where WL and BL represent a word line and a bit line, respectively [55].

The capacitor, which represents one-bit information, is directly driven by the connected bit line, and its electrical charge changes dynamically. When the word line of a DRAM bit cell is activated, the capacitor in the DRAM bit cell is connected to the bit line. A data in the DRAM bit cell can be read out or written by measuring or driving the bit line.

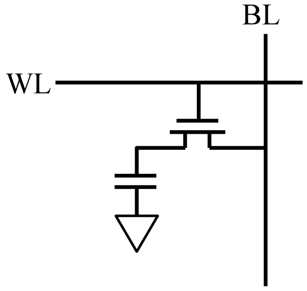


Figure 1.10: DRAM bit cell circuit.

When the data is read out from the DRAM bit cell, the electrical charge in the capacitor is discharged and transferred to the bit line, which eventually breaks the stored one-bit information in the DRAM bit cell. Therefore, the original data in the DRAM bit cell must be rewritten to the DRAM bit cell just after the readout. When the data is written to the DRAM bit cell, the capacitor is directly driven by the bit line. Additionally, since the electrical charge in the capacitor gradually discharges even if there is no readout, a DRAM bit cell must be rewritten, which is called refresh. Typically, a DRAM bit cell has to be refreshed every two or three milliseconds.

Compared to an SRAM bit cell, a DRAM bit cell is slower and has lower throughput due to the slow electrical charge transfer and periodical refresh operations. On the other hand, as shown in Figure 1.10, a DRAM bit cell only requires one transistor and one capacitor to store one bit, whereas an SRAM bit cell requires six transistors, which means that DRAM bit cells are more area efficient than SRAM bit cells. Therefore, there is a trade-off between the area efficiency and the memory access latency, which is reflected in the price of the memory devices and determines where each memory type is applied.

1.5.3 Emerging memory types

SRAMs and DRAMs have been the mainstream semiconductor-based memory types for random access memory (RAM) devices. There are also emerging memory types for non-volatile RAM (NVRAM) devices such as spin-torque transfer magnetic random access memory (STT-MRAM) [57], phase change memory (PCM) [58], resistive random access memory (ReRAM) [59], ferroelectric random access memory (FeRAM) [60], and nanotube random access memory (NRAM) [61].

Since these emerging memory types are studied and developed for NVRAM devices that hold the stored data even if there is no power supply, they are targeted for embedded devices such as wearable devices and storage devices, where power efficiency and data stability are inevitable. However, some of these memory types also aim to replace the main memory of general-purpose computers. In particular, it is reported that STT-MRAM and NRAM have the potential to replace DRAM in terms of memory access latency, area efficiency, and power efficiency [61–66].

1.6 Content addressable memory (CAM)

CAM is a special type of memory device that returns the memory address as a result of searching operations based on the input data. Figure 1.11 shows a high-level structure of CAM that has $\boldsymbol{P'}$ words, each consists of $\boldsymbol{Q'}$ bits, where ML and SL represents the search line and match line, respectively [67]. Typically, CAMs have 36 to 144 bits per word and several hundreds of kilo words.

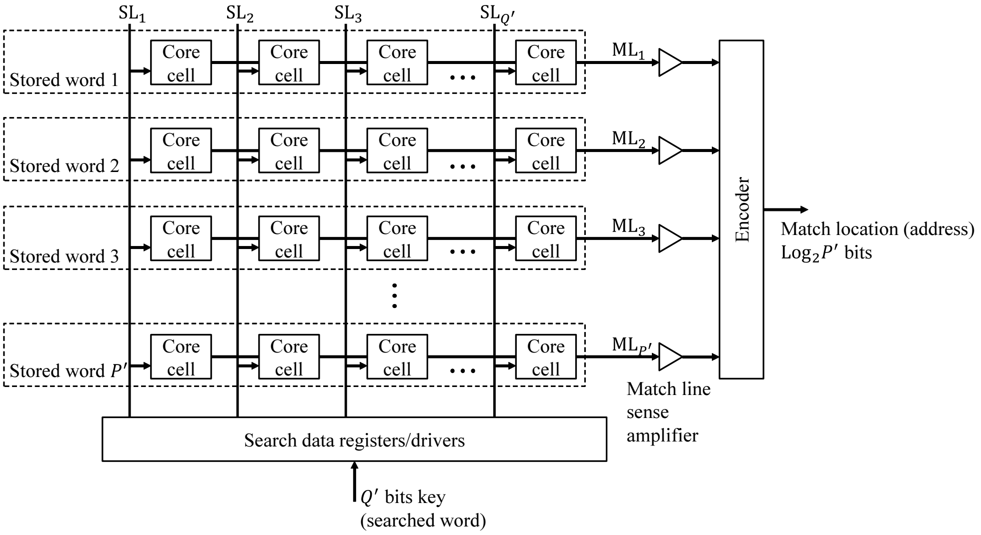


Figure 1.11: High-level structure of CAM.

Each word has Q' CAM core cells, each of which contains one bit of the word and a comparison circuit. Each bit of an incoming key is broadcasted to CAM core cells through the corresponding search line. In each CAM core cell, the incoming bit is compared to the stored bit to judge if the incoming bit equals the stored bit. The horizontal array of CAM core cells are connected via a match line; each horizontal array corresponds to the stored word. In each horizontal array of CAM core cells, the comparison result of each CAM core cell is aggregated by calculating "and" logic via the a match line, which means that the match line eventually indicates if the incoming key is identical to the stored word or not. In other words, if the match line is high, it indicates that the match line is in a matched state; otherwise, it is in a mismatch state. The comparison results in each match line are sensed and amplified separately and transferred to the encoder that outputs the location (address) of the matched key in the database stored in the CAM. Therefore, the incoming key is simultaneously compared to all the words stored in the CAM in one clock cycle.

TCAM is a special type of CAMs, which have "don't care" bit in each core cell in addition to 0 and 1. The "don't care" bit does not contribute to the state of the match line regardless of the input key bit. The existence of "don't care" bit in a stored word corresponds to masking some bits of the searched word, which is required in LPM of IP address lookup, complex string lookup. In comparison with TCAMs, normal CAMs are also called binary CAMs (BCAMs).

In the searching operation, all the match lines are initially pulled up and set in a matched state. Then the search word is broadcasted to all the CAM core cells via search lines. In each CAM core cell, the broadcasted bit is compared with the stored bit. By taking and of all the comparison results of each CAM core cell in the same match line, including the "don't care" bits, the final state of the match line is determined. If there is any mismatch in at least one CAM core cell in the match line, the match line is eventually in the mismatch state, which is pulled down to indicate the mismatch. If there is no mismatch in the match line, the match line remains activated, which indicates the matched state.

The difference between TCAMs and BCAMs is the method to prioritize the multiple matched lines. Figure 1.12 shows an example implementation of IP address lookup subsystem based on a TCAM and a RAM [1, 68].

In the TCAM shown in the left-hand side of Figure 1.12, the searching key 01011 is compared with all the stored words in the TCAM. Considering the “don’t care” bits, denoted as X in Figure 1.12, it is clear that both ML_2 and ML_3 are to be in the matched state. In TCAM that has “don’t care” bits, the encoder can more flexibly choose the final match location from the multiple matched lines based on such as the increasing or decreasing order of the address of the word, the longest matched prefix, or the number of matched CAM bit cells. Even in BCAMs, multiple matched lines can be prioritized based on the increasing or decreasing order of the address of the word. However, BCAMs cannot prioritize the multiple matched lines based on the longest prefix or number of matched CAM bit cells, which is not applied to complex searching operations such as LPM in the IP address lookup task.

In Figure 1.12, the encoder of the TCAM chooses the ML_2 as the final match location based on the longest prefix match. Then the searching results of the TCAM are transferred to the RAM that is connected to the TCAM to get the forwarding port of the router and next hop IP address [68]. When using TCAMs for LPM of IP address lookup, all the routing prefixes must be stored in the TCAM in decreasing order of their prefix length, with “don’t care” bits padded in the rightmost side of the stored words.

Due to its dedicated architecture, CAMs can execute over hundreds of millions of searches per second. However, the architecture is completely different from the normal types of memories, in which the data is returned according to the address input, which means that CAMs are the dedicated memory devices solely for the searching operations. In addition, CAMs require large silicon area to implement the CAM cells and logic, which degrade the area efficiency compared to other types of memories and increases the device price. Additionally, since the searching operation is executed in all the CAM core cells simultaneously, CAMs consume a large amount of power and produce lots of heat, which increases the operational cost.

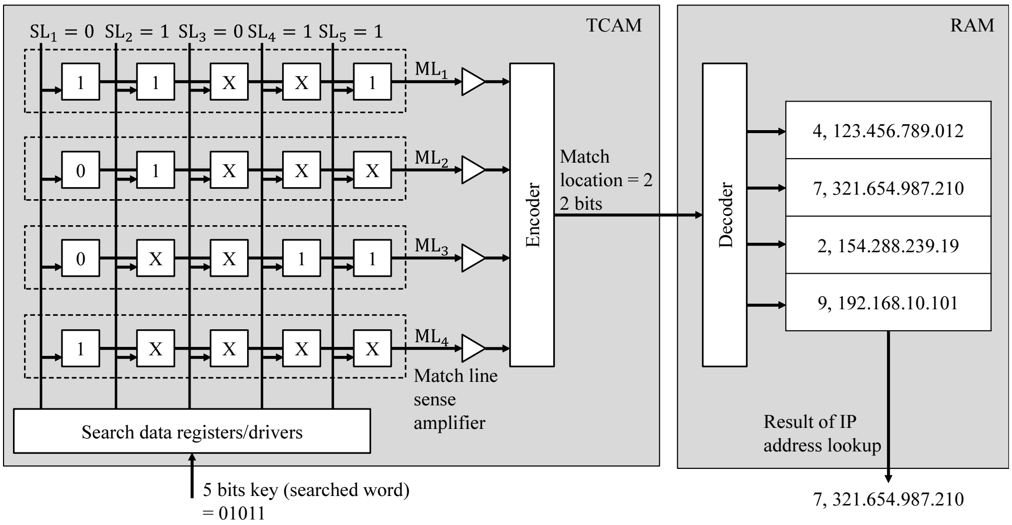

SL₁ = 0 SL₂ = 1 SL₃ = 0 SL₄ = 1 SL₅ = 1
TCAM
1 1 X X 1
0 1 X X X
0 X X 1 1
1 X X X X
ML₁
ML₂
ML₃
ML₄
Match line sense amplifier
Encoder
Match location = 2 2 bits
Search data registers/drivers
5 bits key (searched word) = 01011
RAM
Decoder
4, 123.456.789.012
7, 321.654.987.210
2, 154.288.239.19
9, 192.168.10.101
Result of IP address lookup
7, 321.654.987.210

Figure 1.12: TCAM-based IP address lookup subsystem.

1.7 Memory systems in general-purpose computers

1.7.1 Main memory system

The main memory systems in general-purpose computers are usually based on the DRAM memory system. The DRAM memory system consists of a memory controller and memory devices, as shown in Figure 1.13. The memory controller handles memory requests from requestors, such as CPUs or direct memory accesses (DMAs), to read the data from memory devices or write the data to memory devices. The memory controller logic is usually integrated inside the CPUs. The memory controller and memory devices are connected by a command bus and a data bus. Both buses are accessible in parallel, which means that one requestor can use the command bus while another requestor uses the data bus at the same time. However, no more than one requestor can use the same bus simultaneously.

Modern DRAM systems have a dual inline memory module (DIMM) interface with multiple channels, which allows requestors to access multiple DIMMs simultaneously using multiple command bus and data bus units. Note that multiple DIMMs might be attached to a channel to share the buses in the chan-

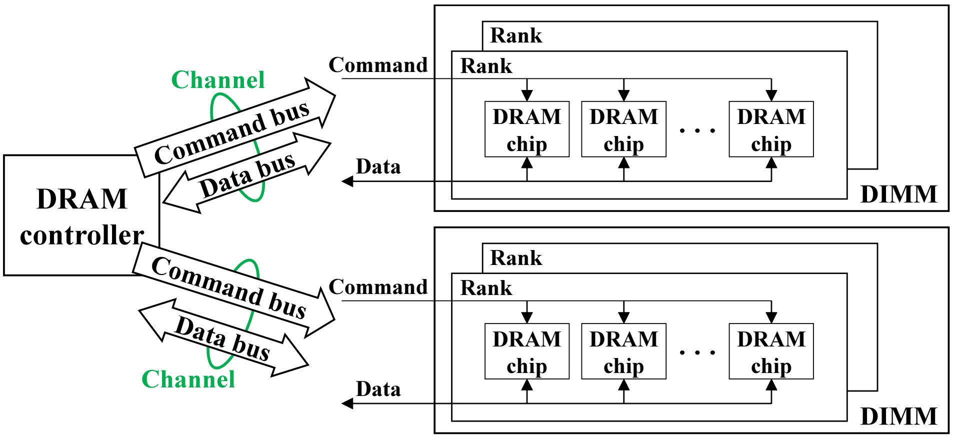


Figure 1.13: DRAM system overview.

nel among the DIMMs, which means that the DIMMs attached to the channel cannot be accessed at the same time. A DIMM is organized into ranks, and only one rank can be accessed at a time. This thesis considers DRAMs with a single rank in the proposed architectures for simplicity.

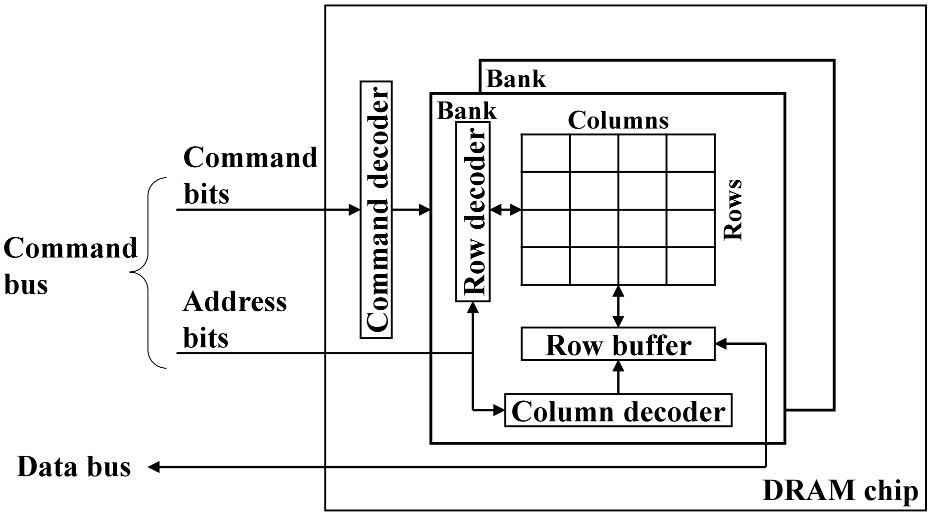
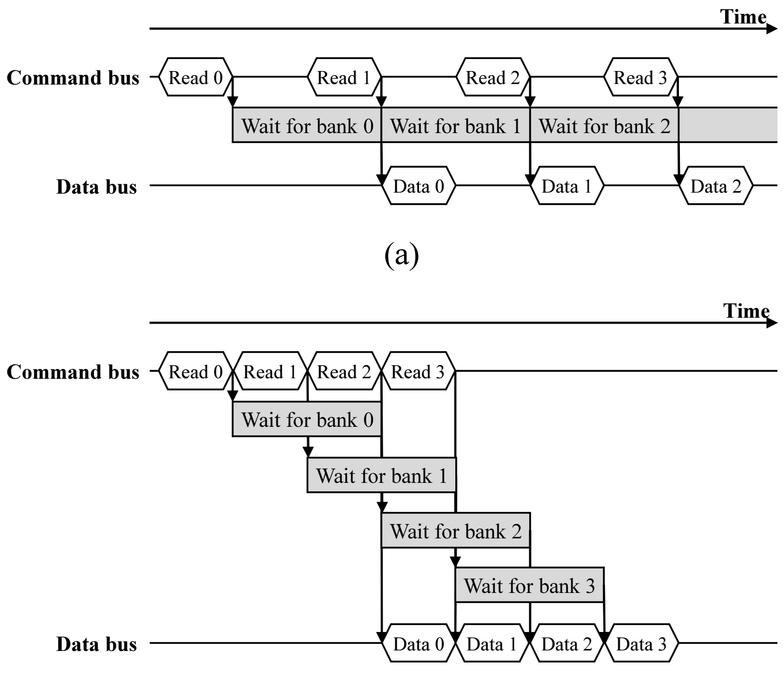


Figure 1.14: Architecture of DRAM chip.

Each rank consists of multiple DRAM chips. Furthermore, each DRAM chip comprises multiple banks that can be accessed in parallel if there are no collisions on either bus. Each bank has a row buffer and a DRAM memory array shown in Figure 1.14. Requestors can only access the content of the row buffer, not the data in the memory array. To access a specific memory location, the row that contains the desired data must be loaded into the row buffer by an activate command. When the controller wishes to load a different row, the current row buffer must be written back to the array by a precharge command in advance. The actual read or write commands only handle the data in the row buffer. A row that is cached in the row buffer is usually referred to as an open row. On the other hand, a row that is not cached in the row buffer is considered as a closed row.

Figure 1.15 shows a schematic diagram of DRAM bank interleaving. Figures 1.15 (a) and (b) correspond to the diagram without bank interleaving and with bank interleaving, respectively. By issuing read commands to an open row at one bank to another, these banks can be interleaved to increase memory access performance with no additional hardware modification.



(a)

(b)

Figure 1.15: DRAM accessing without and with bank interleaving.

1.7.2 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 private cache memories, usually called level 1 (L1) cache memory and level 2 (L2) cache memory. The LLC is shared among every CPU core inside the same CPU. Usually, the L1 cache memory is the fastest and has the smallest memory capacity; on the other hand, LLC is the slowest with the largest memory capacity.

Figure 1.16 shows a basic mechanism of a cache memory system including off-chip main memory. 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 1.16, is likely to be accessed next, which is known as data spatial locality. By assuming the data spatial locality, cache lines improve the hit probabilities of cache memories.

If a cache line including the target data that corresponds to a memory request is found in a certain level of cache memory, which is called a hit, the cache memory 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 memory, which is called a miss, the request accesses the next level of cache memory 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 memory. 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 memory or the L2 cache memory is evicted to the L2 cache memory or the LLC, and the LRU cache line in the LLC is dropped.

Modern CPUs have the on-chip LLC that comprises distributed slices, each of which belongs to each CPU core, as shown in Figure 1.17. Every CPU core including its LLC slice is connected via an inter-core bus that is usually a bi-directional ring or mesh architecture [69,70]. According to [71], Intel's Skylake generation processor 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

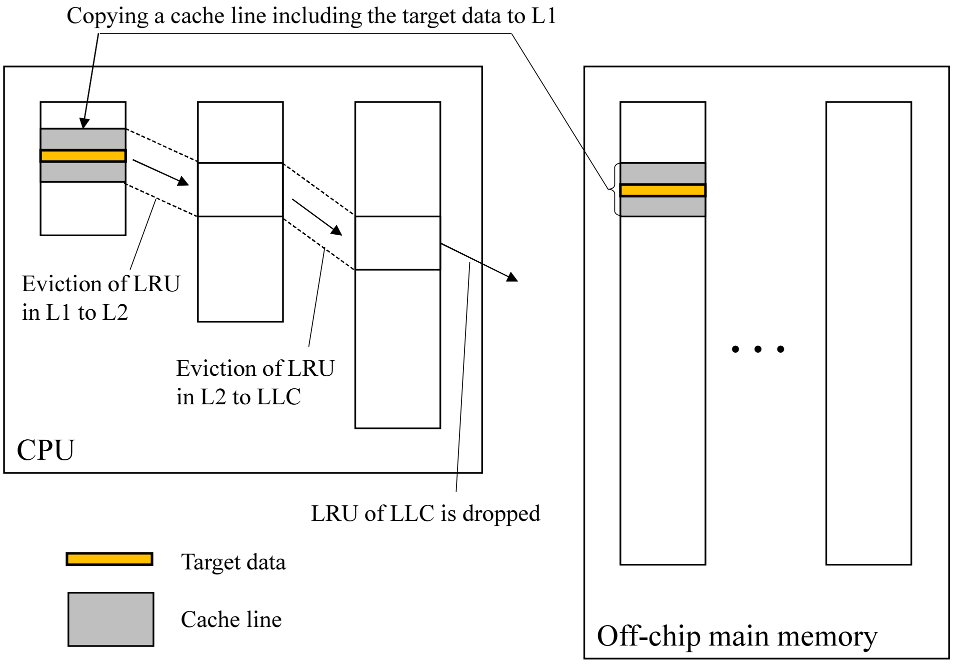

Copying a cache line including the target data to L1
Eviction of LRU in L1 to L2
Eviction of LRU in L2 to LLC
CPU
LRU of LLC is dropped
Target data
Cache line
Off-chip main memory
• • •

Figure 1.16: Basic mechanism of cache memory systems.

2D mesh interconnect among CPU cores and LLC slices of each CPU core was presented in [72]. Thus, although the details of the microarchitecture of such CPUs are not publicly known, the work in [72] considered that today's multi-core CPU uses 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 [73].

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

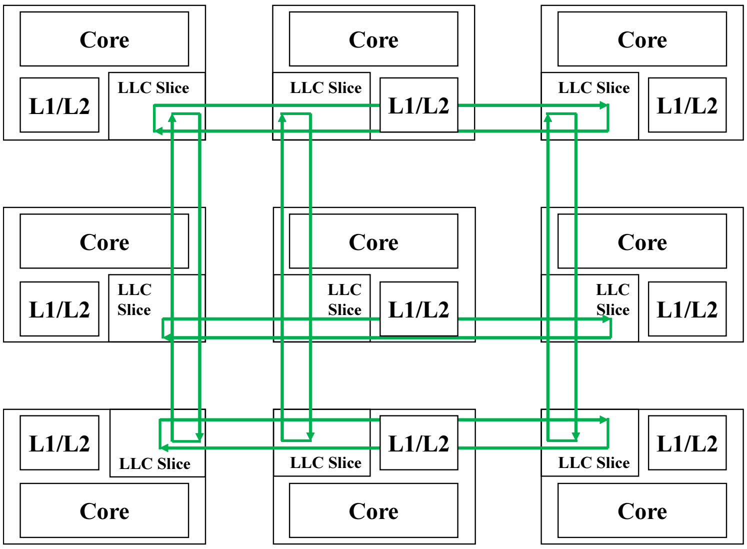


Figure 1.17: Architecture of LLC that consist of LLC slice connected via mesh inter-core bus. Inter-core bus is shown in green lines.

each LLC slice can be evenly balanced. While the detail of the hash function is usually undocumented and not open to the public, 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 [74, 75]. Figure 1.18 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 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 memory is transferred.

In the NFV environment, various applications may 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 which are not assigned to the application. These applications are usually called noisy neighbors as they consume extra LLC slices in which the other applications

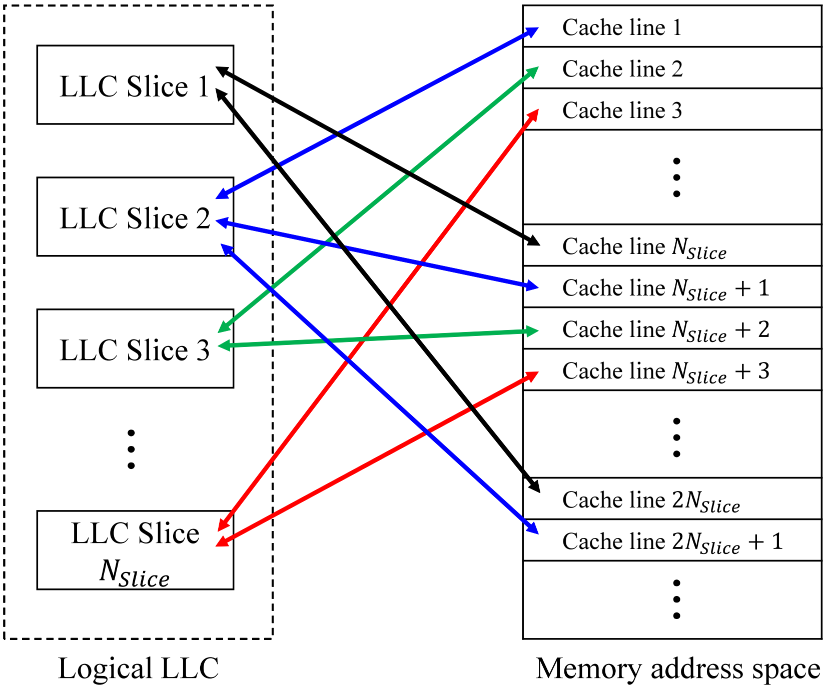


Figure 1.18: Example of address mapping to LLC slices.

are to allocate. Regarding this problem, LLC slices are also becoming one of the major 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) [76] are becoming popular in the operation of NFVI [74].

1.7.3 Three-dimensional (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 x (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 due to the limited wiring space in general-purpose server chassis. The higher density of memory channels in the 3D-stacked DRAMs enables CPU to be connected with more number of memory channels than using conventional DDRx DRAM devices. Thus 3D-stacked DRAM provides more memory channels without losing the versatility of conventional DDRx DRAM devices, which becomes the motivation to use a 3D-stacked DRAM as a parallel main memory in the proposed architectures.

A hybrid memory cube (HMC) [77, 78] and a high bandwidth memory (HBM) [79,80] are example types of 3D-stacked DRAMs. 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 1.19 shows the structure of an HMC. An HMC comprises several DRAM layers on top of the bottom layer, the logic base [81]. Each DRAM layer of a vault has several banks. A vault, which is a vertical unit that consists of a part of each DRAM layer with several banks, corresponds to a memory channel in traditional DRAM devices. Each vault is accessible in parallel. In the logic base, simple operations against the data stored in the DRAM layers can be performed, which is used in emerging computing architecture, known as computing near memory or processing-in-memory (PIM) [82–84]. The external memory controller of an HMC is connected to the logic base of the HMC through several high-speed serial links. The high-speed serial links are also used to connect multiple HMCs.

Figure 1.20 shows the schematic structure of an HBM. As with HMCs, an HBM has several core dies (DRAM layers), each of which comprises conventional DRAM connected through TSV. An HBM has a wider I/O bus compared to an HMC to enhance the memory bandwidth. The bottom layer of an HBM is called base logic die [79].

The thermal feasibility of memory systems using 3D-stacked DRAMs was studied in [85–87]. These works consider the PIM architectures, which produce more heat than conventional DRAM systems due to the aggressive use of the logic layer functionalities. In particular, a challenge is that the heat generated from the logic layer raises the temperature of the DRAM layers. The typical operating temperature range for DRAM is under 85 °C. When the

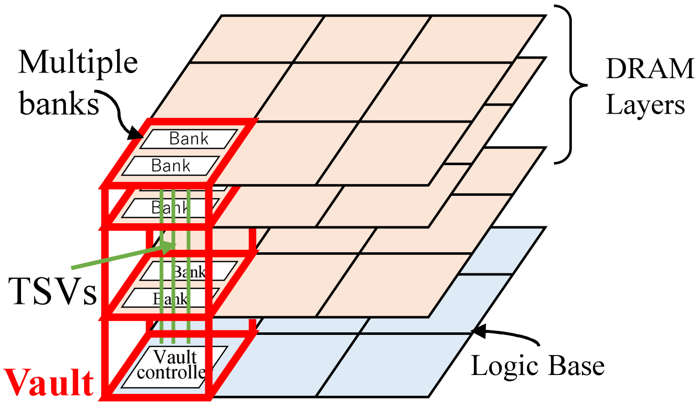


Figure 1.19: Structure of hybrid memory cube.

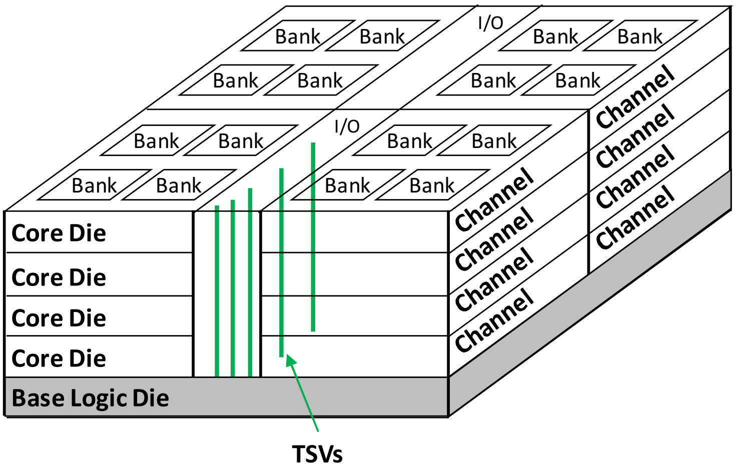


Figure 1.20: Structure of high bandwidth memory.

temperature exceeds the threshold, the required DRAM refresh rate must be doubled for every 10 °C increase [88]. Higher DRAM refresh rates result in higher power consumption and DRAM performance degradation. These works concluded that 3D-stacked DRAMs are feasible if the systems have high-end active cooling, which is typically used in conventional computer systems.

Note that this thesis uses the word "3D-stacked DRAM" in a broad sense, which means a memory device that has a logic layer, several serial interconnects, and several DRAM layers with several channels and banks, and is not limited to HMCs and HBMs.

1.8 Challenges of packet processing in network virtualization

As explained in Section 1.2, while network virtualization is expected to bring cost reduction of network infrastructure and network operation, there are still performance issues to be addressed in order to satisfy the network service requirements when general-purpose computers are used to run VNFs with packet processing. These issues limit the type of network functions to which network virtualization can be applied, which makes it difficult for service providers and telecom operators to benefit from network virtualization.

Every network function includes packet processing tasks, each of which contains various types of operations, as described in Section 1.1. In particular, packet processing tasks such as classification, searching, queueing, and modification issue memory requests to read or write the memory to understand the packet information, update the packet content, and search the databases.

The rate of performance improvement in microprocessor has been higher than that in memory over recent decades [2–8], which increases the importance of memory performance in packet processing. In other words, the more memory accesses are completed in a certain period of time, the more packets are processed in the network equipment or computer systems on which network functions run. Due to the memory bottleneck, conventional computer architectures have increased the memory performance by using parallel memories rather than depending on the slow increase in per-device memory speed [50].

Additionally, several works have presented research directions in computer architectures such as memory-centric computing and 3D-stacking for further performance improvement [89–92].

Conventional network equipment uses dedicated computer architecture and purpose-built hardware devices to increase packet processing performance. While its main processor may be a general-purpose, multi-core processor, its memory system consists of several function-dedicated memories, and has memory parallelism, where TCAMs are used to search the databases quickly and some parallel SRAMs and DRAMs are used to execute packet modification tasks and queueing and traffic management tasks in parallel.

In network virtualization, network functions with packet processing tasks are implemented as software on general-purpose computers whose computer architectures are significantly different from that of dedicated network equipment. Apart from dedicated network equipment, general-purpose computers do not have the function-dedicated memories. While general-purpose computers have processors with several tens of cors, the number of memory channels are much less than the number of processor cores. VNFs on top of general-purpose computers are more likely to wait for the completion of memory accesses, which eventually degrade throughput and increase waiting time. Therefore, it is challenging to increase packet processing performance, eventually the VNF performance, on general-purpose computers.

1.9 Problem statements

This thesis studies four problems about parallel memory system architectures for packet processing in network virtualization. Each problem corresponds to the main memory parallelism, integration of the on-chip cache memories of the CPU with the parallel main memory, capacity and parallelism of the on-chip cache memories in the presence of parallel main memory, and accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses, respectively.

1.9.1 Memory parallelism in main memory

There are concerns about lack of packet processing performance of VNFs running on general-purpose computers. As described in Section 1.3.1, in conventional network equipment, high-performance packet processing is achieved by parallel processing based on dedicated computer architecture and purpose-built hardware devices. However, general-purpose computers have neither such architectures nor purpose-built chips that have dedicated instruction set architectures, which means that general-purpose computers are not originally suitable for U-plane processing in network virtualization.

While there are various approaches to accelerate packet processing in network virtualization such as Intel Data Plane Development Kit (DPDK) [93], Single Root I/O Virtualization (SR-IOV) [94], and software switches and routers [53, 95–98], they mainly increase packet I/O performance or significantly depend on cache memory performance of the CPU with limited memory capacity, which cannot be applied to every packet processing. Moreover, in service providers and telecom networks, each VNF usually comprises various complicated packet processing tasks, each of which has large databases for large traffic flows or complicated functions such as DPI. Therefore, for virtualization of such large-scale networks, off-chip main memory performance needs to be increased for packet processing. There is a lack of memory parallelism in conventional main memory systems in general-purpose computers, where the number of concurrently accessible memory blocks is much less than the increasing number of CPU cores in multi-core CPUs.

Therefore, the problem is to design the parallel main memory system architecture of general-purpose computers to increase packet processing performance with large databases in network virtualization. This thesis studies this problem in Chapter 3.

1.9.2 Integration of on-chip cache memories with parallel main memory

CPUs in general-purpose computers have on-chip cache memories to obscure the latency of the main memory. Since there is usually a trade-off between memory response speed and capacity, modern multi-core CPUs have several levels of on-chip cache memories, which contain the L1 cache memory that is the fastest but smallest, the L2 cache memory that has medium speed and capacity, and the LLC that is the slowest but the largest. Usually, the LLC is shared among all the CPU cores on the same chip. Software-based high-performance switches and routers make the most of these on-chip cache memories of CPUs by limiting the implemented packet processing tasks and compressing the database for packet processing [14, 99].

Since the problem stated in Section 1.9.1 considers the parallel main memory system architecture of general-purpose computers, the existence of on-chip cache memories of the CPU is not taken into account. Even if there is a parallel main memory in general-purpose computers, on-chip cache memories of CPU are also fundamental components of memory systems. Therefore, it is inevitable to explore the integration of the on-chip cache memories with the parallel main memory, in which each level of cache memory has different characteristics, for packet processing in network virtualization. In particular, the effectiveness of the on-chip cache memories is not clear when integrated with the parallel main memory that improves packet processing performance, rather than conventional main memory that has little memory parallelism.

Therefore, the problem is to explore the integration of the on-chip cache memories with the parallel main memory of general-purpose computers for packet processing in network virtualization. This thesis studies this problem in Chapter 4.

1.9.3 Capacity and parallelism of on-chip cache memories in presence of parallel main memory

While the on-chip shared LLC in the CPU has the largest memory capacity, it lacks the memory parallelism, which may be a performance bottleneck when there is an off-chip parallel main memory. On the other hand, the on-chip shared LLC comprises LLC slices, each of which belongs to each CPU cores like the L1/L2 cache memories, via a ring bus or a mesh interconnection among the CPU cores. Therefore, the LLC slices can be accessed in parallel and increase the per-core capacity of cache memories.

While the problem stated in Section 1.9.2 explores the integration of the on-chip cache memories including the shared LLC with the off-chip parallel main memory, capacity and parallelism of the LLC slices are not considered. There is a technology to assign some of the LLC slices to a specific application running on the general-purpose computers [76], as described in Section 1.7.2. The work in [74] introduces the data replication among the LLC slices to increase the effective memory bandwidth in parallel processing, which means that the LLC slice can be one of the fundamental components in memory systems of general-purpose computers. By using the LLC slices as the independent cache memory that belongs to each CPU core, both memory parallelism of CPU cache memories and the per-core total memory capacity are expected to be increased. It is necessary to understand the dependency of memory performance on the number of assigned LLC slices when combined with the off-chip parallel main memory and the on-chip L1/L2 cache memories of each CPU core.

Therefore, the problem is to design the memory system architecture that integrates the LLC slices with the parallel main memory and the on-chip L1 and L2 cache memories in order to increase capacity and parallelism of the on-chip cache memories for packet processing in network virtualization. This thesis studies this problem in Chapter 5.

1.9.4 Accumulated latency of data transfers between processors and memories

While the problems stated in Sections 1.9.1, 1.9.2, and 1.9.3 consider the parallel memory system architectures to increase packet processing performance on general-purpose computers, there are still data transfers between the processors and the memories for every memory access. The data transfer latency is accumulated when there are multiple memory accesses in packet processing. Moreover, the accumulated latency increases when there are an increasing number of packet processing tasks and necessity to access the memories for multiple times to complete a task, which eventually degrade throughput and latency of VNFs. Regarding the data transfer latency, there are PIM architectures, in which a part of processing is executed in the logic of 3D-stacked DRAMs [5, 82–84, 100, 101]. While PIM architectures reduce the number of data transfers, the majority of processing is still in the processors, which requires a number of data transfers when there are multiple packet processing tasks.

Therefore, the problem is to design the memory system architecture to increase throughput and reduce accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses. This thesis studies this problem in Chapter 6.

1.10 Overview and contributions of this thesis

Figure 1.21 shows the chapter overview of this thesis. Chapter 2 describes the related works.

Chapter 3 proposes a parallel main memory system architecture that uses a 3D-stacked memory for packet processing in network virtualization. The proposed architecture enhances memory parallelism of the main memory of general-purpose computers by leveraging both channel-level parallelism and bank interleaving. In the proposed architecture, the database for packet processing is copied and split into multiple partial databases, each of which is distributed to each set of channel and bank. Incoming memory requests are distributed by a hash-function-based distributor so that the memory requests can concurrently access the corresponding partial database in the main memory. This work also introduces an analytical model of the proposed architecture for two traffic patterns, one with random memory request arrivals and one with bursty arrivals. The results observe that the proposed main memory system architecture increases packet processing performance up to around 80 Gbps for the smallest-sized IP packet involving random and bursty traffic.

Chapter 4 proposes a parallel memory system architecture that integrates the on-chip L1 and L2 cache memories of each CPU core with the off-chip parallel main memory based on a 3D-stacked memory, where frequently accessed entries of the database in the main memory are cached in the on-chip cache memories. This work explores the integration of the on-chip cache memories with the off-chip parallel main memory, where the proposed architecture is compared with two reference architectures, one with the on-chip L1 and L2 cache memories and the on-chip shared LLC, and one without any on-chip cache memories. The results observe that the proposed memory system architecture performs the best in the three architectures, which reduces memory access latency and increases the throughput even in the existence of the main memory parallelism.

Chapter 5 proposes a parallel memory system architecture that integrates the on-chip LLC slices with the off-chip parallel main memory and the on-chip L1 and L2 cache memories to increase capacity and parallelism of on-chip cache memories. In the proposed architecture, a cache line evicted from the on-chip L2 cache memory is stored in one of the corresponding on-chip LLC slices according to a memory-address-based hash function. The proposed architecture also has a skip selection logic to skip the on-chip LLC slices when the LLC miss rate exceeds a threshold. This work compares the proposed architecture with two reference architectures in terms of the integration of the on-chip cache memories with the off-chip 3D-stacked memory: one with the on-chip shared LLC, one without any on-chip cache. The results observe that the proposed memory system architecture outperforms the other architectures in terms of memory access latency and the throughput. Also, the number of assigned LLC slices does not significantly increase memory performance when more than a certain number of LLC slices are assigned. Additionally, the results imply that the skip selection logic can avoid the performance degradation when the size of the database is larger than a certain point, where the LLC miss rate increases.

Chapter 6 proposes a parallel memory system architecture that uses a network of 3D-stacked memory to increase throughput and reduce accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses in network virtualization. The proposed architecture has a memory network, in which incoming packets receive packet processing at each 3D-stacked memory without data transfers between the processors and the memories until the packet processing is completed. Each packet processing task is allocated in the logic of 3D-stacked memories, and the required memory accesses for the task are completed in each 3D-stacked memory. The results observe that the proposed architecture reduces the blocking probability by issuing the next memory request inside the 3D-stacked DRAM just after the previous memory access, instead of allowing the arrivals of incoming packets during the data transfers between the memory and processor. Consequently, the proposed architecture increases the throughput and reduces the accumulated latency when there are multiple packet processing tasks, compared to the architecture with 3D-stacked DRAM-based parallel main memory, where every memory access requires data transfers between the processors and memories. The proposed architecture also reduces the blocking probability and latency by assigning more 3D-stacked DRAMs for a packet processing task that requires more memory accesses than the other tasks.

Finally, Chapter 7 concludes this thesis and discusses the directions for the future works in packet processing in network virtualization.

Chapter 1: background and problem statements

Chapter 2: related works

Parallel memory system architectures
for packet processing in network virtualization

	3D-stacked memory	Private cache	LLC slices	Memory network
Chapter 3	✓	—	—	—
Chapter 4	✓	✓	—	—
Chapter 5	✓	✓	✓	—
Chapter 6	✓	—	—	✓

Chapter 7: conclusions and future works

Figure 1.21: Chapter overview of this thesis.

Chapter 2

Related works

2.1 Packet processing technologies in network virtualization

Several works presented packet processing software on general-purpose computers [14, 36, 53, 95, 96, 102, 103]. Click Modular Router [95] is a software framework for various packet processing, in which a software router can be comprised of several packet processing modules called elements. RouteBricks [96] is the first study that makes the most of the parallelism of modern multi-core CPUs. While the number of CPU cores in general-purpose computers has increased since the work presented, the number of memory channels in general-purpose computers has been constant. Therefore, the parallelism gap between the CPU and the main memory has widened. Lagopus [36,102,103] is a high-performance OpenFlow-compatible software switch that achieves 10 Gbps switching with 1 M flow entries by using DPDK and the on-chip cache memories. While this approach improves packet processing performance compared to the previous packet processing applications running on single-core CPUs, capacity of on-chip cache memories is too small to accommodate multiple and large databases for packet processing, which requires faster, more parallel, and larger memory systems. The proposed architectures increase parallelism of the main memory and cache memories.

GPU is used in order to augment the parallelism of packet processing in [53]. The study assumes that every packet goes through the same processing in order to leverage the GPU's SIMD parallelism. In general, NFV applications process the packets from various users, each of which may have different destinations, priorities, and packet sizes. Thus the SIMD-based parallel packet processing may not be applicable to packet processing in service provider's networks such as telecom carrier networks. Poptrie [14] is a software scheme for high-performance IP routing. While it provides 200 M lookups per second performance on a single CPU core by compressing the lookup tables and by using the popcnt instruction [104] to execute pattern matching efficiently, this software scheme depends on the small on-chip cache memories. Also, since this approach is specialized for the table lookup for IP routing, additional methods are required for other packet processing in general NFV applications. The proposed architectures increase the overall memory performance for various NFV applications rather than a specific application. The proposed architectures also enhance parallelism of the on-chip cache memories and the main memory, which increases packet processing performance whose data size does not suit the small on-chip cache memories.

The work in [97] presented vectorized packet processing, in which a group of packets, called a packet vector, is processed at a time by using vector instructions such as streaming SIMD extensions and Intel advanced vector extensions [104] instead of processing one packet at a time. In the vectorized packet processing, there are ready-made packet processing functions that can be chained to form desired routers. Since it efficiently executes the required instructions for packet processing, it may be combined with the proposed architectures in this thesis to increase the packet processing performance further.

There are several software libraries or frameworks that improve packet I/O performance in general-purpose computers [105,106]. The following approaches are some of the most commonly used ones. Intel DPDK [93] is a hardware-aware framework to increase packet transfer performance between the network interfaces and the VNFs in general-purpose computers by reducing the packet transfer cost of network stacks in OSes. DPDK requires the DPDK-compatible processor architecture, OS, and NIC, whereas they are commodity components in general-purpose computers. DPDK consists of the following technologies: a poll mode driver that enables userspace processes to avoid interrupts and context switching as well as minimize memory copies, Intel direct data I/O that directly copies received packets from NICs to the on-chip cache memories of CPUs, and larger memory pages, called huge pages, to reduce data translation lookaside buffer misses. SR-IOV [94] virtualizes the physical network interfaces and presents the virtualized network interfaces to VMs as if the VMs have the physical network interfaces natively by using PCIe passthrough. Since most of these approaches increase packet I/O performance based on standard hardware devices and OSes in general-purpose computers, they can be integrated with the proposed architectures in this thesis to increase packet processing performance.

Methods to improve packet processing performance in the NFV-aware environment were introduced in [107, 108]. The work in [107] showed that the number of memory copies and the number of memory accesses to remote processors in non-uniform memory access environments significantly affect packet processing performance. The work in [107] also demonstrated that larger table data size degrades packet processing performance due to an increasing number of cache misses as the table data size increases, which leads to the motivation to introduce the 3D-stacked DRAM in the proposed architectures. The work also presented the assignment of resources such as packet queues in NIC hardware and CPU cores to VNFs should be optimized. The work in [108] introduced smart NICs to COTS hardware for NFV. A smart NIC is a NIC that has a network processor or an FPGA so that some part of packet processing in NFV applications can be offloaded to the NIC. In the work, 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 function. These methods to use packet processing accelerators such as smart NICs are compatible with the proposed architecture, which increase the overall memory access performance.

2.2 3D die-stacked memory architectures

In order to address the memory wall problem [2], several architectures that place memory closer to the processor by stacking the memory directly on top of the processor were studied in literature. The works in [109–112] presented the approaches that directly stack DRAM-based cache memories on top of the CPU, which lowers access latency to the DRAM-based cache memories compared to the off-chip DRAMs. In addition, 3D die-stacked DRAM cache extends the capacity of cache memories instead of using small on-chip SRAM-based cache memories. The proposed architectures presented in Chapters 3, 4, and 5 increase the memory parallelism and capacity of cache memories in general-purpose computers using off-chip 3D-stacked main memory and on-chip cache memories. The proposed architecture presented in Chapter 6 eliminates the data transfers between processors and memories using the network of 3D-stacked memories, where each 3D-stacked memory has a programmable logic in the logic die.

The work in [113] presented a die-stacked memory architecture in which not only DRAMs but also NVRAMs such as ReRAMs and NRAMs are considered as the stacked memory. The stacked memory is connected to the on-chip L2 cache memories of each CPU core, which eliminates the I/O pin count constraints and reduces the memory bus latency. Since the main problems of the work in [113] were related to the design of die-stacked memory, the parallelism of the on-chip cache memories and the stacked DRAMs or NVRAMs were not considered extensively. The work considered up to four memory channels of the die-stacked memories, which is no different from the baseline memory parallelism of today's general-purpose computers. The proposed memory system architectures increase parallelism of off-chip main memory and on-chip cache memories, which increases the memory performance.

2.3 Emerging computer architectures

With the emergence of multi-core processor and accelerator hardware devices and the approaching end of Moore's Law [114, 115], several works considered new computer architectures, such as disaggregated computing architectures, memory-centric computing architectures, near memory computing architectures.

The works in [7, 8] presented disaggregated server architectures where processors and memories are disaggregated to address memory wall in terms of memory capacity by scaling them independently. These works introduced compute blades and memory blades, each of which contains a pool of processors and memory modules, respectively. The blades are connected via a backplane. Thus a user can allocate some of the processors and memory capacity from the pooled resources for specific applications and VMs without constrains on memory capacity in physical server hardware. Intel rack scale architecture [116] presented a scalable computer architecture that provides scalability of computer resources such as processors, memories, storage, and network interfaces by assigning the pooled resources on demand. It allows the service provider to scale the computing power within the resources in a rack instead of those in a physical server chassis. The work in [117] introduced software-defined hardware infrastructures, which bring hardware modularity, scalability, and flexibility to virtualized infrastructure for NFV and SDN. It also considered other layers in computer systems, such as networking, infrastructure management, and virtualization platform. Although these approaches provided more flexible, modular, and scalable computer architectures for virtualization, they did not consider packet processing performance of VNFs nor memory parallelism in their computer architectures, which is quite different from the proposed parallel memory system architectures.

The disaggregated computer architectures are categorized as processor-centric computing, in which the processing results are produced inside the processors. On the other hand, there are emerging memory-centric computer architectures such as computing in memory array, computing in peripheral circuitry, and computing near memory [118] to increase memory performance by reducing memory access overhead and the number of data transfers between processors and memories.

The works in [119–122] presented computer architectures that perform computing in memory array. These works use modified bit cells in their memory arrays, each of which can perform simple primitive operations. In this type of architectures, ReRAM-based memory arrays are often considered. The works in [119–121] were based on ReRAM. On the other hand, the work in [122] considered DRAM-based bit cells. The basic structure of these architectures is similar to that of CAMs or TCAMs, in which a bit in the searched word is compared in each bit cell and the result is transferred via a match line. While these architectures are designed for more general-purpose processing, compared to CAMs or TCAMs, which can be used only for searching, they require a redesign of bit cells and their bit lines and word lines to support the additional logic in bit cells. These approaches offload some of the instructions from the CPU to the memory and increase the processing performance by reducing memory access overhead. On the other hand, the proposed memory system architectures enhance the overall memory parallelism for packet processing. Therefore, the architectures for computing memory array are different from the proposed architectures.

The works in [123,124] presented computer architectures that perform computing in memory peripheral circuitry. Since these approaches are independent of the design of bit cells in a memory array, the type of memory arrays are not limited, in which DRAMs, SRAMs, and other NVRAMs can be used. The work in [123] reads multiple memory rows and executes SIMD-like instructions against the multiple rows at the modified sense amplifier of the memory peripheral circuitry, instead of reading and transferring such rows and writing back the computing results from the processor to the memory. It reduces the total number of memory read/write operations, necessary clock cycles, and power consumption. The work in [124] presented data parallel processor, which is a ReRAM-based architecture that leverages instruction- and data-level parallelism for the acceleration of dot-product operations. It also supports other primitive operations such as arithmetic and shift operations, which can be used for general-purpose computing. While architectures for computing in memory peripheral circuitry may be applied to some of the packet processing tasks, they require significant hardware modification inside the memory devices. In contrast, the proposed architectures increases memory performance using general-purpose hardware devices.

There are several works that studied the computer architectures that perform computing near memory, also known as near-memory-computing and PIM. The works in [5, 82, 100, 101] presented architectures for computing near memory based on 3D-stacked memory devices, whose embedded logic circuits are used for the computing. The work in [82] presented an HMC-based architecture that performs large vector operations in the logic die of an HMC. It consists of a host processor and an HMC. Some of the instructions are offloaded from the processor to the logic die in the HMC. The HMC returns the execution results to the host processor. The work in [5] presented an architecture for computing near memory that automatically decides whether to execute an instruction in memory or processors depending on the data locality without changing the existing programming model. The architecture consists of a host processor and an HMC. The work introduced some additional components between the host processor and the HMC to execute some parts of the instructions in the HMC seamlessly. The works in [100, 101] presented architectures that perform computing near memory in the network of HMCs. In the architecture of [100], an in-network computation on the way is conducted in the network of 3D-stacked memory devices. The architecture consists of a host CPU and a memory network that comprises multiple HMCs. In the network of HMCs, HMCs are chained together using the embedded packet switches in the HMCs. The architecture introduces three phases for processing: tree construction phase to create a flow from the processor to the specific HMC that has the required data, update phase for data processing in which each operation against specific data is executed in the same HMC and is chained in the flow, and gather phase for reduction operation. The proposed architecture presented in Chapter 6 uses the network of 3D-stacked memories, where each 3D-stacked memory has a packet processing task, to increase packet processing performance by eliminating the data transfers between processors and memories until the packet processing is completed in the memory network.

2.4 Computer architecture simulation

The emergence of new computer architectures accelerates the demands on the computer architecture simulation to allow researchers to evaluate the new architectures [125,126]. The computer architecture simulation is used to model new ideas for a part of computer systems, such as microprocessor, memory, and I/O system, or an entire computer system in order to estimate the performance improvement in terms of throughput, latency, and power efficiency.

There were several computer architecture simulators, which were used in the works [127–129]. All these simulators support x86 architecture and other major architectures such as ARM architecture. The work in [127] provided gem5 that is an event-driven full-system simulator. It is based on M5 [130] for CPU modeling and GEMS [131] for memory system modeling. It supports several ISA other than x86 and ARM. Out-of-order and simultaneous multithreading are supported in CPU simulation. It can run some part of the target software code that is executed in full-system simulation on the physical hardware in order to accelerate the simulation speed. The work in [128] presented Sniper that is a fast simulator based on the interval simulation method [132]. It is specialized for x86 architecture and supports out-of-order and parallel processing. It also supports on-chip private and shared cache memories, dynamic voltage and frequency scaling, and Intel x86-64 ISA, which makes the Sniper relatively realistic simulator. Additionally, it is officially validated against the Intel Core 2 microarchitecture. The work in [129] presented McSimA+ that simulates x86-based asymmetric many-core microarchitectures, including both core and uncore subsystems. It supports out-of-order. It is positioned between a full-system simulation and an application-level simulation. The simulation is executed in an application-driven way considering the thread management for many-core processing independent from the host OS and physical hardware on which the simulator runs. Hence, the simulation based on McSimA+ can run faster than full-system simulators.

This thesis mainly focuses on the hardware aspect of the parallel memory system architectures in general-purpose computers, which is inevitable to increase the general packet processing performance in network virtualization. Additionally, the proposed memory system architectures use a 3D-stacked DRAM and an FPGA to enhance the main memory parallelism in addition to the standard hardware components in commodity general-purpose servers, whereas the additional devices are general-purpose ones. Therefore, although these computer architecture simulators can model and simulate the newly designed architectures, they do not fit the purpose of performance evaluation of the proposed architecture in this thesis due to the mainly three points described as follows. (1) The computer architecture simulators introduced in this section so far are complicated due to various system parameters and their tuning, multiple options to select the mode of the simulators, and dependence on several third-party software tools to model specific parts of the computer. The complexity of simulators may make it difficult to understand which system parameter gives what performance impact on which part of the architecture. (2) While these simulators can model and simulate the components or entire systems of existing computer architectures, modeling and simulating the additional components, device connections, and system architectures require the modification of the existing simulators and additional development of the simulation software. (3) These simulators are assumed to run the real applications on top of the simulators or to input the traces taken from the real applications. Therefore, the applications or their traces for simulation might be modified so that they can be properly executed on the simulated computer architecture, which may make it difficult to distinguish the performance evaluation of the simulated computer architecture and optimization of the existing application that runs on the simulated computer architecture.

Based on the above consideration, this thesis evaluates performance of the proposed architectures by using analysis and simulations based on queueing models, in which the fundamental memory parallelism of the proposed architectures are modeled independent from implementation of other components in computer systems such as OSes, application types, and memory address translations.

Chapter 3

Parallel memory system architecture using interleaved 3D-stacked memory

This chapter proposes a parallel memory system architecture that uses a 3D-stacked memory to increase the memory parallelism of the main memory for packet processing in network virtualization based on general-purpose hardware. A part of the work in this chapter was presented in [133]. While the on-chip cache memories are effective in fast packet processing in network virtualization, main memory is still a key component of general-purpose computers, in which large databases such as lookup tables for routing traffic of the subscribers and complex packet classification, which cannot fit into the on-chip cache memories, are accommodated. Additionally, lack of memory parallelism in main memory compared to the number of CPU cores is a bottleneck in packet processing with multi-core CPUs in network virtualization.

The proposed architecture enhances memory parallelism by using channel-level parallelism and memory bank interleaving based on 3D-stacked DRAM. In the proposed architecture, the database for packet processing is copied along all the memory channels of the 3D-stacked DRAM, and the database in each memory channel is equally divided into several partial databases, each of which is stored in one of the banks in the memory channel. The proposed architecture uses the hash-function-based distributor that distributes memory requests to an appropriate channel-bank set that has a partial database according to the incoming packet content.

This work introduces analytical models of the proposed architecture for two traffic patterns, one with random memory request arrivals and one with bursty arrivals. This work extensively describes the states and the state transitions, and formulates the equilibrium equations for both memory request arrivals. The numerical evaluation results observe that the proposed architecture increases packet processing performance up to 80 Gbps for the smallest-sized IP packet involving random and bursty memory request arrivals. This work also presents a direction for expanding the analytical model to a general case.

The rest of this chapter is organized as follows. Section 3.1 presents the proposed architecture. Section 3.2 describes the system modeling. Sections 3.3 and 3.4 provide analyses of the proposed architecture for random and bursty request arrivals, respectively. Section 3.5 presents performance evaluations of the proposed architecture. Section 3.6 describes a direction in which to expand the analytical model in this work. Section 3.7 summarizes this chapter.

3.1 Proposed architecture

Figure 3.1 shows the proposed parallel memory system architecture. It consists of a multi-core CPU, an FPGA, a 3D-stacked DRAM, a DRAM and network interfaces. Although the proposed architecture can have multiple CPUs, FPGAs, HMCs, and DRAMs, for simplicity, this work describes and models the proposed architecture as depicted in Figure 3.1.

Each incoming packet is processed as follows.

(Step 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. The assigned CPU core reads the header information of the packet from the packet buffer in the DRAM.

(Step 2) The CPU core issues memory requests to read the database stored in the 3D-stacked DRAM in order to decide the next action for the packet.

(Step 3) After finishing the lookup and determining the next action, the CPU core sends the packet including its payload from the packet buffer in the

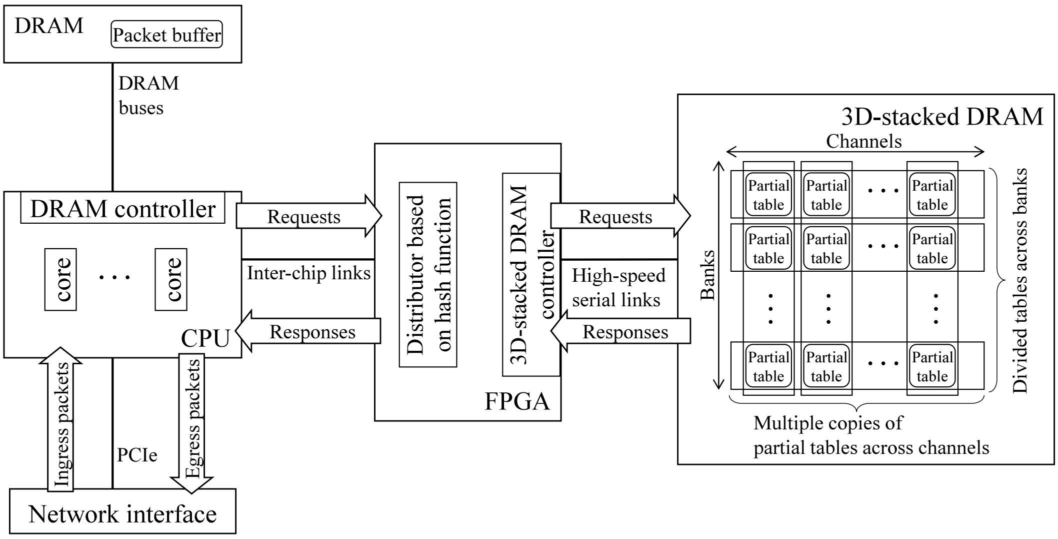


Figure 3.1: Overview of proposed memory system architecture.

DRAM through the network interface.

As explained in Section 2.1, performance of the packet processing steps 1 and 3 can be improved by using the packet I/O acceleration methods such as Intel DPDK. This work focuses on the packet processing step 2 in the above description, which in particular executes the table lookup tasks, as shown in Figure 3.2. The 3D-stacked DRAM holds the database for packet processing such as lookup tables in the following manner in order to leverage both memory channel-level parallelism and memory bank interleaving. In the 3D-stacked DRAM, the database for packet processing is copied along all the memory channels. In every memory channel, the database is equally divided into several partial databases so that the original database in each memory channel comprises the partial databases in the memory channel. The number of partial databases equals the number of banks in each memory channel, and the number of copies equals the total number of memory channels in the 3D-stacked DRAM.

The 3D-stacked DRAMs have much more memory channels than the conventional DRAMs. For example, according to HMC specifications [81], an HMC has up to 32 memory channels per device; however, an Intel Xeon CPU of Skylake generation has only six memory channels [134]. In addition, the

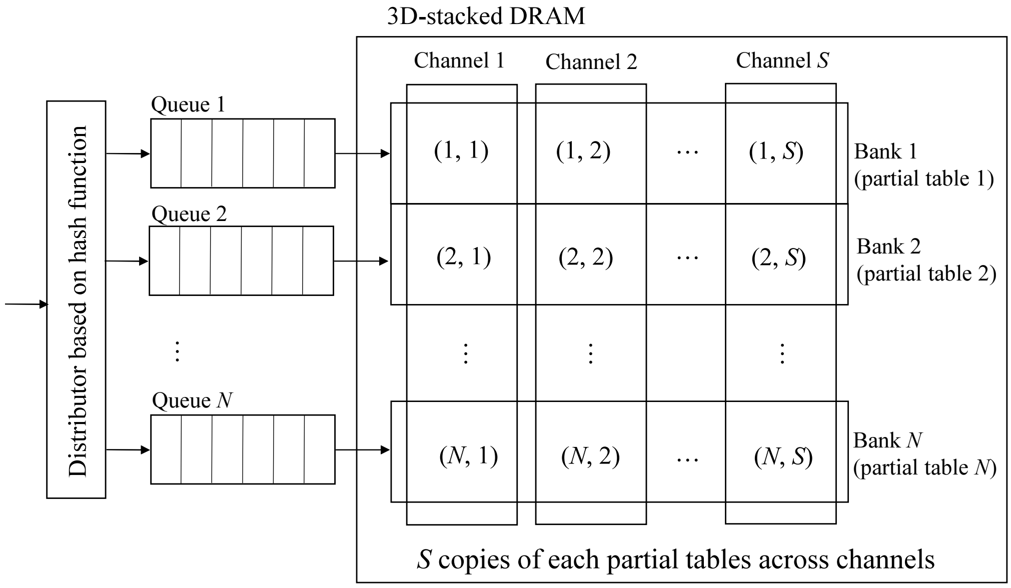


Figure 3.2: Main memory system of the proposed architecture.

conventional main memory system architecture has little room to increase the number of memory channels for DRAM devices due to the complex electrical wiring for the DRAM buses between a CPU and DRAM devices. Therefore, the proposed architecture has an advantage in terms of memory channel-level parallelism over the conventional main memory system architecture, in which distributed databases in the 3D-stacked DRAM can be simultaneously accessed by multiple CPU cores. In addition to memory bank interleaving, this advantage increases the packet processing performance in network virtualization.

An FPGA is used to connect the CPU to the 3D-stacked DRAM. The hash-function-based distributor circuit is deployed in the FPGA. According to the incoming packet content, the distributor distributes the memory requests to an appropriate channel-bank set of the 3D-stacked DRAM that contains the corresponding partial database of the packet 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. The memory controller logic of the 3D-stacked DRAM is also deployed in the FPGA, which interfaces the 3D-stacked DRAM.

Several semiconductor companies have already released the intellectual property core products of the 3D-stacked DRAM controller for FPGAs [135–139]. 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 [140]. This FPGA-based product may also 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. Since LUTs in FPGAs are based on SRAMs, the processing latency of FPGAs is sufficiently smaller than the DRAM access latency. Therefore the memory parallelism and bandwidth are more critical metrics than the additional latency introduced by the FPGA. Thus this work considers that an FPGA is a candidate to connect the 3D-stacked DRAM and the CPU. Recent CPUs have inter-chip links such as UPI, which can be used to connect the CPU and the FPGA [141–143].

In the proposed architecture, memory requests are served simultaneously by using multiple memory channels of 3D-stacked DRAM. This may change the order of egress packets from the processor, which affects the performance of upper layer such as transmission control protocol (TCP) [144]. In order to eliminate the misordered packets, there are several approaches: to exchange signals among multiple processes or threads so that every packet can be served in order and to buffer the packets and sort them before transmitted from the processor [145, 146].

This work assumes that dividing the database for packet processing into partial databases is conducted based on the database structure. For example, the works in [15, 147, 148] consider dividing the original database into several partial databases in order to make the database more memory efficient so that the database fits within a specific memory capacity.

This work also assumes that the database accommodated in the 3D-stacked DRAM is in a stable condition, in which there is no database update such as routing table updates. This work assumes that the database in the 3D-stacked DRAM is updated using the corresponding functions of a specific database structure, if necessary. For example, database update schemes are considered in several database structures such as [13, 14, 147]. Additionally, since the database is replicated and placed in every memory channel in the proposed architecture, the databases are updated sequentially for each memory channel.

3.2 System model

This section describes the behavior of the main memory system in the proposed architecture shown in Figure 3.2. The 3D-stacked memory device accommodates tables such as IP routing tables. The proposed architecture consists of a distributor based on hash function, N queues, and the 3D-stacked DRAM. The 3D-stacked DRAM consists of S memory channels, each memory channel has N banks. In every memory channel, a whole lookup table is separated into N partial tables, each of which is allocated to one of the banks in the memory channel. S copies of every partial table across memory channel are made; each memory channel has the same table entries.

When a memory request enters the distributor, the hash function in the distributor classifies the request to one of N queues by using packet information, such as destination IP address. The calculation in this hash function is as simple as to classify the result of an logical operation for some bits of packet header, which is usually finished in one clock cycle in FPGA. Requests entering queue n, where $n \in [1, N]$, are served in a first come first served (FCFS) manner. Queue n has S servers, each of which corresponds to a memory channel. The sth server for queue n, where $s \in [1, S]$, is denoted by server (n, s). The maximum number of requests that can be accommodated, including all the queues and servers, is K, where $K \geq NS$. A request entering the main memory system is blocked if the number of requests already being handled by the main memory system is K. The packet generating the blocked request is discarded. The memory resources are shared by N queues under the condition that the total number of accommodated requests in the main memory system does not exceed K. In the worst case, $K - S$ requests are waiting for service in one particular queue and S requests are served by the corresponding S servers. The memory access rate by queue n to bank n at memory channel s is the service rate of server (n, s); each server serves one request. When more than one server, each of which corresponds to a different bank, at memory channel s are active, or more than one request is being served, bank interleaving is performed among them. Otherwise, no bank interleaving is performed.

When bank interleaving is performed using w banks at memory channel s, this work calls the interleaving w-degree bank interleaving; no bank interleaving is performed when $w = 1$.

This work describes the analytical model. Each server is in one of three states, idle, busy without bank interleaving, and busy with bank interleaving. A server in idle state does not serve any request. A server in busy state without bank interleaving serves a request without bank interleaving. A server in busy state with bank interleaving serves a request with bank interleaving. When at least a server in idle state for queue n exists, a request at the head of line is served in the following server selection rule. If there is any server in idle state that moves to busy state without bank interleaving, it is selected. Otherwise, a server in idle state that moves to busy state with bank interleaving that has the least degree of interleaving, is selected.

Figure 3.3 shows a state transition diagram for each server. When sever (n, s) in idle state serves a request, the state moves to busy state with w-degree bank interleaving so as to minimize the value of w. When server (n, s) in busy state with w-degree bank interleaving finishes serving a request and does not serve any request, it enters idle state. When server (n', s) in idle state starts to serve a request, server (n, s) $(n \neq n')$ in busy state with w-degree bank interleaving moves to busy state with $(w + 1)$-degree bank interleaving. When server (n', s) finishes serving a request and does not serve any new request, server (n, s) $(n \neq n')$ in busy state with w-degree bank interleaving enters busy state with $(w - 1)$-degree bank interleaving.

This work assumes that a request arrives at the main memory system following a Poisson arrival process with average rate of λ, and the distributor based on a hash function distributes the request among N queues. Therefore, a request is assumed to arrive at each queue based on a Poisson arrival process with average rate of $\frac{\lambda}{N}$. Since the processing time of DRAM depends on the DRAM mechanism, in which precharge is required before loading another row, this work assumes that the service rate of server (n, s) follows an exponential distribution with average service rate of μ_w for w-degree bank interleaving, where $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_N$ with $w\mu_w \geq \mu_1$.

This work focuses on analyzing the case for $N = 2$, as it is the simplest case that includes bank interleaving. This work builds two queueing models to analyze the performance of proposed architecture under two types of traffic models. For each traffic model, this work describes all feasible states of system with the proposed architecture and analyze the transitions between them with considering the case of $N = 2$

Figure 3.4 shows a state transition diagram for each server. When server (n, s) in idle state serves a request, the state moves to busy state with or without bank interleaving. When server (n, s) in busy state with bank interleaving finishes serving a request and does not serve any new request, it enters idle state. When server (n', s) in idle state starts to serve a request, server (n, s) $(n \neq n')$ in busy state without bank interleaving moves to busy state with bank interleaving. When server (n', s) finishes serving a request and does not serve any new request, server (n, s) $(n \neq n')$ in busy state with bank interleaving moves to busy state without bank interleaving.

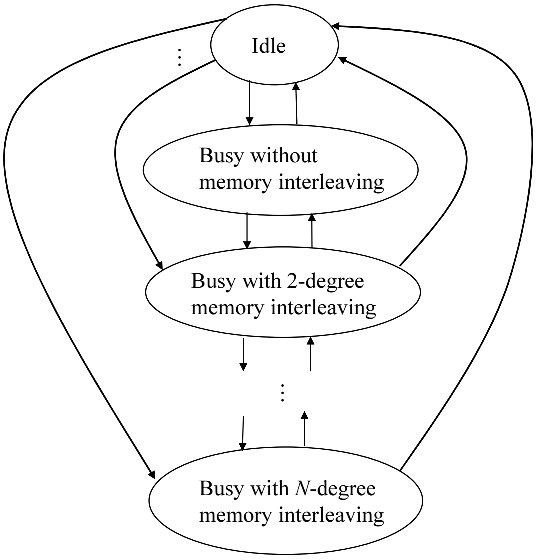

Idle
Busy without memory interleaving
Busy with 2-degree memory interleaving
Busy with N-degree memory interleaving

Figure 3.3: State transition for each server.

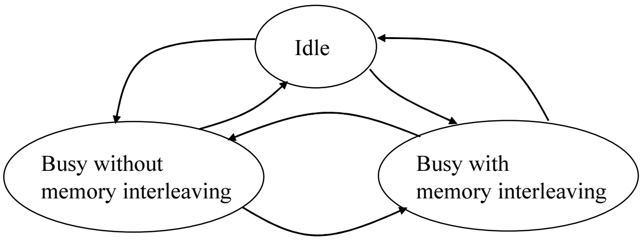

Idle
Busy without memory interleaving
Busy with memory interleaving

Figure 3.4: State transition for each server in $N = 2$.

3.3 Analysis for random arrival of memory requests

3.3.1 States description

This work describes the analytical model of the proposed architecture with $N = 2$ to analyze its performance. Since a Markov process for request arrivals and service times with and without bank interleaving is assumed, a state in the main memory system is expressed by (i, j, p), where $i \in [0, K]$ is the number of requests for bank 1, $j \in [0, K]$ is the number of requests for bank 2, and $p \in [0, S]$ is the number of requests being served with 2-interleaving for both banks. The service rates for requests being served without memory interleaving and with 2-interleaving are different. There can be some states with the same (i, j) but different p, for each of which the outgoing transfer rates to the states with $(i-1, j)$ or $(i, j-1)$ due to the termination of service of a request depend on the corresponding number of requests being served with 2-interleaving for both banks. Therefore, p is required to be included to identify a state. Since the memory resources are shared by queues 1 and 2, $i+j \leq K$ must be satisfied. Let X denote $[0, K]$.

This work describes all possible feasible states to derive the number of states. The states are divided into three cases for the values of i and j, two of which are further divided to several sub cases with considering the range of p. In case 1, i, j, and S are not equal to each other. In case 2, only two of them are equal. In case 3, all of them are equal. γ_1, γ_2 and γ_3 denote the number of feasible states for case 1, case 2, and case 3, respectively. Γ denotes the total number of feasible states in the main memory system, where $\Gamma = \gamma_1 + \gamma_2 + \gamma_3$.

In case 1, i, j, and S are not equal to each other $(i \neq j, i \neq S, j \neq S)$. As the range of p depends on i, j, $i+j$, and S, case 1 is divided into three sub cases, case 1a, case 1b, and case 1c, which depend on the range of i. γ_1^a, γ_1^b, and γ_1^c denote the number of feasible states for case 1a, case 1b and case 1c, respectively.

In case 1a, $i \in [0, \lfloor S/2 \rfloor]$, where the symbol of $\lfloor x \rfloor$ denotes the maximum integer that does not exceed x. When $i = 0$, $j \in (0, S) \cup (S, K]$ and $p = 0$ are obtained. Therefore, there are $(S-1) + (K-S) = K-1$ feasible states in this situation. When $i \in [1, \lfloor S/2 \rfloor]$, the range of j is $[0, i) \cup (i, S-i] \cup (S-i, S) \cup (S, K-i]$. If $j \in [0, i)$, which means $j < i < S$ and $i + j \leq S$, $p \in [0, j]$ is obtained, which has $j+1$ feasible states for each j. Therefore, there are $\sum_{j=0}^{i-1}(j+1) = \sum_{t=1}^{i} t$ feasible states in total for each i when $i \in [1, \lfloor S/2 \rfloor]$ and $j \in [0, i)$. If $j \in (i, S-i]$, which means $i < j < S$ and $i + j \leq S$, $p \in [0, i]$ is obtained, which has $i+1$ feasible states for each j, where there are $S-i-i$ possibilities. Therefore, there are $(S-2i)(i+1)$ feasible states in total for each i when $i \in [1, \lfloor S/2 \rfloor]$ and $j \in (i, S-i]$. If $j \in (S-i, S)$, which means $i < j < S$ and $i + j > S$, $p \in [i+j-S, i]$ is obtained, which yields $S-j+1$ feasible states for each j. Therefore, there are $\sum_{j=S-i+1}^{S-1}(S-j+1) = \sum_{t=2}^{i} t$ feasible states in total for each i when $i \in [1, \lfloor S/2 \rfloor]$ and $j \in (S-i, S)$. If $j \in (S, K-i]$, which means $i < S < j$, $p = i$. Therefore, there are $K-S-i$ feasible states in total for each i when $i \in [1, \lfloor S/2 \rfloor]$ and $j \in (S, K-i]$. As a result, by summing all of the number of feasible states for each $i \in [0, \lfloor S/2 \rfloor]$, the total number of feasible states for case 1a is given by

$$\gamma_1^a = K - 1 + \sum_{i=1}^{\lfloor S/2 \rfloor} \left[\sum_{t=1}^{i} t + (S-2i)(i+1) + \sum_{t=2}^{i} t + (K-S-i) \right]. \tag{3.1}$$

In case 1b, $i \in [\lfloor S/2 \rfloor + 1, S)$ and the range of j is $[0, S-i] \cup (S-i, i) \cup (i, S) \cup (S, K-i]$. If $j \in [0, S-i]$, which means $j < i < S$ and $i + j \leq S$, $p \in [0, j]$ is obtained, which has $j+1$ feasible states for each j. Therefore, there are $\sum_{j=0}^{S-i}(j+1) = \sum_{t=1}^{S-i+1} t$ feasible states in total for each i when $i \in [\lfloor S/2 \rfloor + 1, S)$ and $j \in [0, S-i]$. If $j \in (S-i, i)$, which means $j < i < S$ and $i + j > S$, $p \in [i+j-S, j]$ is obtained, which yields $S-i+1$ feasible states for each j, where there are $i-(S-i)-1 = 2i-S-1$ possibilities. Therefore, there are

$(2i-S-1)(S-i+1)$ feasible states in total for each of i when $i \in [\lfloor S/2 \rfloor+1, S)$ and $j \in (S-i, i)$. If $j \in (i, S)$, which means $i < j < S$ and $i+j > S$, $p \in [i+j-S, i]$ is obtained, which has $S-j+1$ feasible states for each j. Therefore, there are $\sum_{j=i+1}^{S-1}(S-j+1) = \sum_{t=2}^{S-i} t$ feasible states in total for each i when $i \in [\lfloor S/2 \rfloor+1, S)$ and $j \in (i, S)$. If $j \in (S, K-i]$, which means $i < S < j$, $p = i$. Therefore, there are $K-S-i$ feasible states in total for each i when $i \in [\lfloor S/2 \rfloor+1, S)$ and $j \in (S, K-i]$. As a result, by summing all of the number of feasible states for each $i \in [0, \lfloor S/2 \rfloor]$, the total number of feasible states for case 1b is given by

$$\gamma_1^b = \sum_{i=\lfloor S/2 \rfloor+1}^{S-1} \left[\sum_{t=1}^{S-i+1} t + (2i-S-1)(S-i+1) + \sum_{t=2}^{S-i} t + (K-S-i) \right]. \tag{3.2}$$

Case 1c is divided into the four cases of $i \in (S, \lfloor K/2 \rfloor]$, $i \in (\lfloor K/2 \rfloor, K-S)$, $i = K-S$, and $i \in (K-S, K]$. If $i \in (S, \lfloor K/2 \rfloor]$, $j \in [0, S) \cup (S, i) \cup (i, K-i]$[1], and there are $K-i-1$ possibilities of j. If $i \in (\lfloor K/2 \rfloor, K-S)$, $j \in [0, S) \cup (S, K-i]$, and there are $K-i$ possibilities of j. If $i = K-S$, $j \in [0, K-i)$, and there are $K-i$ possibilities of j. If $i \in (K-S, K]$, $j \in [0, K-i]$, and there are $K-i+1$ possibilities of j. As a result, the total number of feasible states for case 1c is given by

$$\gamma_1^c = \sum_{S+1}^{\lfloor K/2 \rfloor} (K-i-1) + \sum_{\lfloor K/2 \rfloor+1}^{K-S} (K-i) + \sum_{K-S+1}^{K} (K-i+1). \tag{3.3}$$

Therefore, the total number of feasible states for case 1 is given by

$$\gamma_1 = \gamma_1^a + \gamma_1^b + \gamma_1^c. \tag{3.4}$$

In case 2, only two of them are equal. There are six sub cases, which are $i = j < S$ for case 2a, $S < i = j$ for case 2b, $i < j = S$ for case 2c, $j = S < i$ for case 2d, $j < i = S$ for case 2e, and $i = S < j$ for case 2f. γ_2^a, γ_2^b, γ_2^c, γ_2^d, γ_2^e, and γ_2^f denote the number of feasible states for case 2a, case 2b, case 2c, case 2d, case 2e, and case 2f, respectively. In case 2a, if $i \in [0, \lfloor S/2 \rfloor]$, which means $i+j \leq S$, $p \in [0, i]$ is obtained, which yields $i+1$ feasible states for each i, so there are $\sum_{t=0}^{\lfloor S/2 \rfloor}(t+1)$ feasible states. If $i \in [\lfloor S/2 \rfloor+1, S)$, which means $i+j > S$, $p \in [2i-S, i]$ is obtained, which yields $S-i+1$ feasible states for each

[1] When K is an even number and $i = K/2$, $(i, K-i]$ is $(i, i]$, which is an empty set.

i, so there are $\sum_{t=\lfloor S/2\rfloor+1}^{S-1}(S-t+1)$ feasible states. Therefore, the number of feasible states for case 2a is given by $\gamma_2^a = \sum_{t=0}^{\lfloor S/2\rfloor}(t+1) + \sum_{t=\lfloor S/2\rfloor+1}^{S-1}(S-t+1)$. In case 2b with $S < i = j$, clearly, $p = S$ if $i \in (S, \lfloor K/2\rfloor]$. Therefore, $\lfloor K/2\rfloor - S$ feasible states can be obtained for case 2b, or $\gamma_2^b = \lfloor K/2\rfloor - S$. In case 2c with $i < j = S$, clearly, $p = i$ if $i \in [0, S)$. Therefore, S feasible states are obtained for case 2c, or $\gamma_2^c = S$. In case 2d with $j = S < i$, clearly, $p = S$ if $i \in (S, K - S]$. Therefore, $K - 2S$ feasible states are obtained for case 2d, or $\gamma_2^d = K - 2S$. In case 2e with $j < i = S$, clearly, $p = j$ if $j \in [0, S)$. Therefore, S feasible states are obtained for case 2e, or $\gamma_2^e = S$. In case 2f with $i = S < j$, clearly, $p = S$ if $j \in (S, K - S]$. Therefore, $K - 2S$ feasible states are obtained for case 2f, or $\gamma_2^f = K - 2S$. As a result, γ_2 is given by

$$\begin{aligned}\gamma_2 &= \gamma_2^a + \gamma_2^b + \gamma_2^c + \gamma_2^d + \gamma_2^e + \gamma_2^f \\ &= \sum_{t=0}^{\lfloor S/2\rfloor}(t+1) + \sum_{t=\lfloor S/2\rfloor+1}^{S-1}(S-t+1) + \lfloor K/2\rfloor + 2K - 3S. \qquad (3.5)\end{aligned}$$

In case 3, all of i, j and S are equal ($i = j = S$). p is always equal to S and there is just one feasible state for case 3, $\gamma_3 = 1$.

Therefore, by summing all the number of states for each case, the total number of feasible states in the main memory system is given by,

$$\begin{aligned}\Gamma &= \gamma_1 + \gamma_2 + \gamma_3 \\ &= \sum_{i=1}^{\lfloor S/2\rfloor}\left[\sum_{t=1}^{i} t + (S-2i)(i+1) + \sum_{t=2}^{i} t + (K-S-i)\right] \\ &\quad + \sum_{i=\lfloor S/2\rfloor+1}^{S-1}\left[\sum_{t=1}^{S-i+1} t + (2i-S-1)(S-i+1) + \sum_{t=2}^{S-i} t + (K-S-i)\right] \\ &\quad + \sum_{S+1}^{\lfloor K/2\rfloor}(K-i-1) + \sum_{\lfloor K/2\rfloor+1}^{K-S}(K-i) + \sum_{K-S+1}^{K}(K-i+1) \\ &\quad + \sum_{t=0}^{\lfloor S/2\rfloor}(t+1) + \sum_{t=\lfloor S/2\rfloor+1}^{S-1}(S-t+1) + \lfloor K/2\rfloor + 3K - 3S. \qquad (3.6)\end{aligned}$$

Based on the discussion on feasible states in the main memory system, the range of p is obtained as $p \in [\min(\max(0, i+j-S), i, j, S), \min(i, j, S)]$. Let $Y(i, j)$ denote $[\min(\max(0, i+j-S), i, j, S), \min(i, j, S)]$.

3.3.2 State transition for (i, j, p)

Figure 3.5 shows the state transitions incoming to and outgoing from state (i, j, p), where eight states are incoming to and eight states are outgoing from state (i, j, p). Table 3.1 describes the rate and condition for each transition, where each case is numbered from 1 to 16.

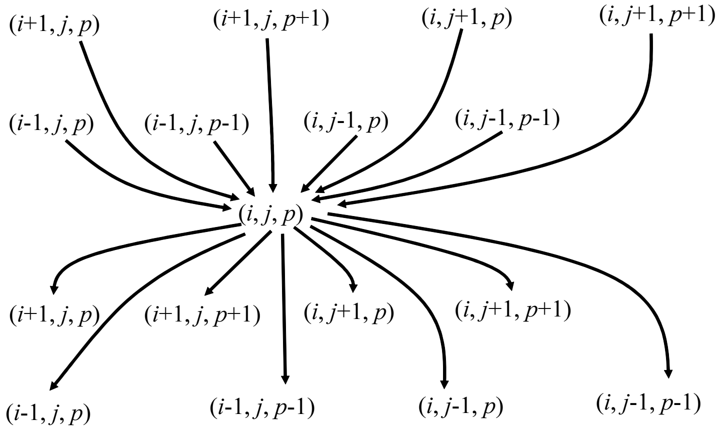


Figure 3.5: State transitions incoming to and outgoing from state (i, j, p).

3.3.3 Equilibrium states

Let $P(i, j, p)$ be the probability that the main memory system is in state (i, j, p). Let U be the set of states (i, j, p), where $i \in X$, $j \in X$, and $p \in Y(i, j)$. In the equilibrium state, the total incoming flows to state (i, j, p) are equal to the total outgoing flows from state (i, j, p). The equilibrium equations for $(i, j, p) \in U$ are given by,

$$\begin{aligned}
&(q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 + q_8)P(i, j, p) \\
&= q_9 P(i-1, j, p) + q_{10} P(i-1, j, p-1) + q_{11} P(i, j-1, p) \\
&\quad + q_{12} P(i, j-1, p-1) + q_{13} P(i+1, j, p) + q_{14} P(i+1, j, p+1) \\
&\quad + q_{15} P(i, j+1, p) + q_{16} P(i, j+1, p+1), \qquad (3.7)
\end{aligned}$$

where q_c, $c \in [1, 16]$, equals the transfer rate of case c if the conditions of case c are satisfied and 0 otherwise.

Table 3.1: State transitions incoming to and outgoing from state (i, j, p).

Direction	Case	State	Transfer rate	Condition
Incoming states	9	$(i-1, j, p)$	$\lambda/2$	$(0 \leq i-1 < S$ and $S-(i-1+j-p) > 0)$ or $i-1 \geq S$
	10	$(i-1, j, p-1)$	$\lambda/2$	$S-(i+j-p) \leq 0$ and $0 \leq i-1 < S$ and $p > 0$
	11	$(i, j-1, p)$	$\lambda/2$	$(0 \leq j-1 < S$ and $S-(i+j-1-p) > 0)$ or $j-1 \geq S$
	12	$(i, j-1, p-1)$	$\lambda/2$	$S-(i+j-p) \leq 0$ and $0 \leq j-1 < S$ and $p > 0$
	13	$(i+1, j, p)$	$(i+1-p)\mu_1$	$i+1 \leq S$ and $(i+1+j) \leq K$
			$(S-p)\mu_1 + p\mu_2$	$S < i+1 \leq K$ and $(i+1+j) \leq K$
	14	$(i+1, j, p+1)$	$(p+1)\mu_2$	$i+1 \leq S$ and $(i+1+j) \leq K$
	15	$(i, j+1, p)$	$(j+1-p)\mu_1$	$j+1 \leq S$ and $(i+j+1) \leq K$
			$(S-p)\mu_1 + p\mu_2$	$S < j+1 \leq K$ and $(i+j+1) \leq K$
	16	$(i, j+1, p+1)$	$(p+1)\mu_2$	$j+1 \leq S$ and $(i+j+1) \leq K$
Outgoing states	1	$(i-1, j, p)$	$(i-p)\mu_1$	$i \leq S$
			$(S-p)\mu_1 + p\mu_2$	$i > S$
	2	$(i, j-1, p)$	$(j-p)\mu_1$	$j \leq S$
			$(S-p)\mu_1 + p\mu_2$	$j > S$
	3	$(i-1, j, p-1)$	$p\mu_2$	$0 < i \leq S$ and $p > 0$
	4	$(i, j-1, p-1)$	$p\mu_2$	$0 < j \leq S$ and $p > 0$
	5	$(i+1, j, p)$	$\lambda/2$	$(i < S$ and $S-(i+j-p) > 0$ and $(i+1+j) \leq K)$ or $(S \leq i < K$ and $(i+1+j) \leq K)$
	6	$(i+1, j, p+1)$	$\lambda/2$	$S-(i+j-p) \leq 0$ and $i < S$ and $(i+1+j) \leq K$
	7	$(i, j+1, p)$	$\lambda/2$	$(j < S$ and $S-(i+j-p) > 0$ and $(i+j+1) \leq K)$ or $(S \leq j < K$ and $(i+j+1) \leq K)$
	8	$(i, j+1, p+1)$	$\lambda/2$	$(S-(i+j-p) \leq 0$ and $j < S$ and $(i+j+1) \leq K)$

The condition that the sum of all state probabilities equals one is given by,

$$\sum_{(i,j,p)\in U} P(i,j,p) = 1. \tag{3.8}$$

The probability of each state $P(i,j,p) \in U$ can be computed by solving the multiple equations of (3.7) and (3.8).

3.3.4 Blocking probability and average waiting time

This work defines the blocking probability $P_{\mathrm{b}}^{\mathrm{R}}$ as the probability that a request incoming to the main memory system is blocked with the condition of $i+j = K$, or the request is not able to enter the queue. $P_{\mathrm{b}}^{\mathrm{R}}$ is given by

$$P_{\mathrm{b}}^{\mathrm{R}} = \sum_{i\in X} \sum_{p\in Y(i,j)} P(i,K-i,p). \tag{3.9}$$

This work defines the average waiting time at the main memory system, W^{R}, as the average duration time from when a request enters the main memory system until the request exits the main memory system. The average number of requests in the main memory system, L^{R}, is given by

$$\begin{aligned} L^{\mathrm{R}} &= \sum_{i\in X}\sum_{j\in X}\sum_{p\in Y(i,j)} iP(i,j,p) + \sum_{i\in X}\sum_{j\in X}\sum_{p\in Y(i,j)} jP(i,j,p) \\ &= 2\sum_{i\in X}\sum_{j\in X}\sum_{p\in Y(i,j)} iP(i,j,p). \end{aligned} \tag{3.10}$$

The first/second terms on the right hand side of the first equality indicate the average number of requests waiting at queue 1/queue 2 and those being served, respectively. The right hand side for the second equality is derived by using $\sum_{i\in X}\sum_{j\in X}\sum_{p\in Y(i,j)} iP(i,j,p) = \sum_{i\in X}\sum_{j\in X}\sum_{p\in Y(i,j)} jP(i,j,p)$.

By using Little's formula [149],

$$W^{\mathrm{R}} = \frac{L^{\mathrm{R}}}{\lambda}. \tag{3.11}$$

Let $\lambda_{\mathrm{e}}^{\mathrm{R}}$ be the throughput, which is defined by

$$\lambda_{\mathrm{e}}^{\mathrm{R}} = \lambda\left(1 - P_{\mathrm{b}}^{\mathrm{R}}\right). \tag{3.12}$$

Let $W_{\mathrm{e}}^{\mathrm{R}}$ be the average effective average waiting time, which is defined by,

$$W_{\mathrm{e}}^{\mathrm{R}} = \frac{L^{\mathrm{R}}}{\lambda_{\mathrm{e}}^{\mathrm{R}}}. \tag{3.13}$$

3.4 Analysis for bursty arrival of memory requests

This work adopts the Interrupted Poisson Process (IPP) as the packet arrival process to analyze the main memory system under bursty traffic conditions in the steady state. For the transient analysis of the queueing systems, several works presented analytical models for specific problems [150, 151] and methods to reduce the amount of computation [152, 153]. In particular, the work in [151] studied the transient solutions in statistical multiplexing. The statistical multiplexing is used in the UDP and TCP protocols, where the traffic of data stream with variable packet length is shared among multiple channels identified by the port number. The works in [154, 155] use machine learning technologies to optimize the control of queueing systems.

3.4.1 Overview of IPP

There are two states as regards the arrival of memory requests, ON and OFF states. The durations of ON and OFF states follow exponential distributions with average inter-arrival times $\frac{1}{\alpha}$ and $\frac{1}{\beta}$, respectively. Once the ON state finishes, the OFF state starts, and vice versa. The ON state includes a Poisson arrival process of requests with average rate λ. There is no request arriving at the main memory system when the state of the arrival of requests is OFF. The state transition diagram of IPP is shown in Figure 3.6.

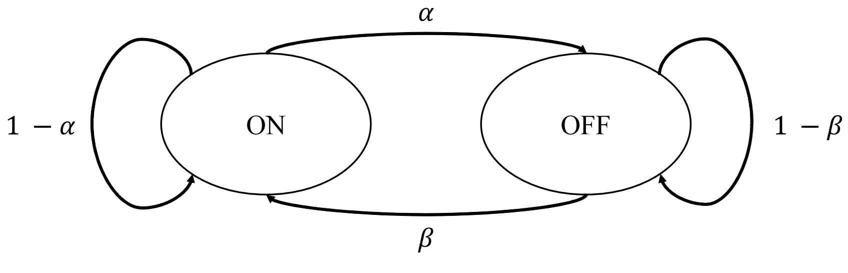


Figure 3.6: State transition diagram of IPP.

This work assumes that, in the ON state, requests are consecutively destined to the same bank until the ON state finishes. Let $k \in [0, N]$ denote the

Table 3.2: State transitions incoming to and outgoing from state $(i, j, p, 0)$ in IPP.

Direction	Case	State	Transfer rate	Condition
Incoming states	16	$(i+1, j, p, 0)$	$(i+1-p)\mu_1$	$i+1 \leq S$ and $(i+1+j) \leq K$
			$(S-p)\mu_1 + p\mu_2$	$S < i+1 \leq K$ and $(i+1+j) \leq K$
	17	$(i+1, j, p+1, 0)$	$(p+1)\mu_2$	$i+1 \leq S$ and $(i+1+j) \leq K$
	18	$(i, j+1, p, 0)$	$(j+1-p)\mu_1$	$j+1 \leq S$ and $(i+j+1) \leq K$
			$(S-p)\mu_1 + p\mu_2$	$S < j+1 \leq K$ and $(i+j+1) \leq K$
	19	$(i, j+1, p+1, 0)$	$(p+1)\mu_2$	$j+1 \leq S$ and $(i+j+1) \leq K$
	21	$(i, j, p, 1)$	α	
	22	$(i, j, p, 2)$	α	
Outgoing states	1	$(i-1, j, p, 0)$	$(i-p)\mu_1$	$i \leq S$
			$(S-p)\mu_1 + p\mu_2$	$i > S$
	2	$(i, j-1, p, 0)$	$(j-p)\mu_1$	$j \leq S$
			$(S-p)\mu_1 + p\mu_2$	$j > S$
	3	$(i-1, j, p-1, 0)$	$p\mu_2$	$0 < i \leq S$ and $p > 0$
	4	$(i, j-1, p-1, 0)$	$p\mu_2$	$0 < j \leq S$ and $p > 0$
	10	$(i, j, p, 1)$	$\beta/2$	
	11	$(i, j, p, 2)$	$\beta/2$	

state of the arrival of a packet; k is set to $n \in [1, N]$ when it is ON state in which the packet is destined to bank $n \in [1, N]$, and zero otherwise. Consequently, for the main memory system with $N = 2$, a state in the main memory system is expressed as (i, j, p, k).

The total number of feasible states of (i, j, p, k) is 3Γ. Equation (3.6) gives Γ, which is the total number of feasible states of (i, j, p). In IPP, for each (i, j, p), there are three states where $k = 0, 1, 2$.

3.4.2 State transition for (i, j, p, k)

Figure 3.7 shows the state transitions incoming to and outgoing from states $(i, j, p, 0)$, $(i, j, p, 1)$, and $(i, j, p, 2)$. Tables 3.2, 3.3 and 3.4 describe the rates and conditions for states $(i, j, p, 0)$, $(i, j, p, 1)$, and $(i, j, p, 2)$, respectively, where each transition case is numbered from 1 to 22.

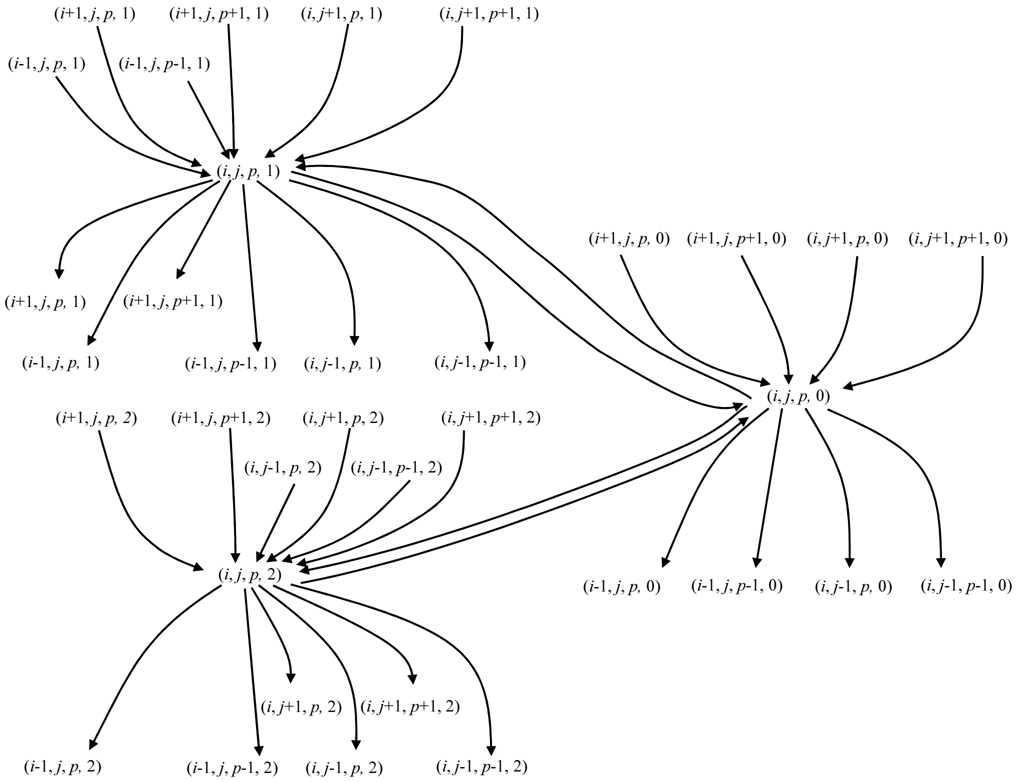


Figure 3.7: State transitions incoming to and outgoing from states $(i, j, p, 0)$, $(i, j, p, 1)$, and $(i, j, p, 2)$.

Table 3.3: State transitions incoming to and outgoing from state $(i, j, p, 1)$ in IPP.

Direction	Case	State	Transfer rate	Condition
Incoming states	12	$(i-1, j, p, 1)$	λ	$(0 \le i-1 < S$ and $S-(i-1+j-p) > 0)$ or $i-1 \ge S$
	13	$(i-1, j, p-1, 1)$	λ	$S-(i+j-p) \le 0$ and $0 \le i-1 < S$ and $p > 0$
	16	$(i+1, j, p, 1)$	$(i+1-p)\mu_1$	$i+1 \le S$ and $(i+1+j) \le K$
			$(S-p)\mu_1 + p\mu_2$	$S < i+1 \le K$ and $(i+1+j) \le K$
	17	$(i+1, j, p+1, 1)$	$(p+1)\mu_2$	$i+1 \le S$ and $(i+1+j) \le K$
	18	$(i, j+1, p, 1)$	$(j+1-p)\mu_1$	$j+1 \le S$ and $(i+j+1) \le K$
			$(S-p)\mu_1 + p\mu_2$	$S < j+1 \le K$ and $(i+j+1) \le K$
	19	$(i, j+1, p+1, 1)$	$(p+1)\mu_2$	$j+1 \le S$ and $(i+j+1) \le K$
	20	$(i, j, p, 0)$	$\beta/2$	
Outgoing states	1	$(i-1, j, p, 1)$	$(i-p)\mu_1$	$i \le S$
			$(S-p)\mu_1 + p\mu_2$	$i > S$
	2	$(i, j-1, p, 1)$	$(j-p)\mu_1$	$j \le S$
			$(S-p)\mu_1 + p\mu_2$	$j > S$
	3	$(i-1, j, p-1, 1)$	$p\mu_2$	$0 < i \le S$ and $p > 0$
	4	$(i, j-1, p-1, 1)$	$p\mu_2$	$0 < j \le S$ and $p > 0$
	5	$(i+1, j, p, 1)$	λ	$(i < S$ and $S-(i+j-p) > 0$ and $(i+1+j) \le K)$ or $(S \le i < K$ and $(i+1+j) \le K)$
	6	$(i+1, j, p+1, 1)$	λ	$S-(i+j-p) \le 0$ and $i < S$ and $(i+1+j) \le K$
	9	$(i, j, p, 0)$	α	

Table 3.4: State transitions incoming to and outgoing from state $(i, j, p, 2)$ in IPP.

Direction	Case	State	Transfer rate	Condition
Incoming states	14	$(i, j-1, p, 2)$	λ	$(0 \le j-1 < S$ and $S-(i+j-1-p) > 0)$ or $j-1 \ge S$
	15	$(i, j-1, p-1, 2)$	λ	$S-(i+j-p) \le 0$ and $0 \le j-1 < S$ and $p > 0$
	16	$(i+1, j, p, 2)$	$(i+1-p)\mu_1$	$i+1 \le S$ and $(i+1+j) \le K$
			$(S-p)\mu_1 + p\mu_2$	$S < i+1 \le K$ and $(i+1+j) \le K$
	17	$(i+1, j, p+1, 2)$	$(p+1)\mu_2$	$i+1 \le S$ and $(i+1+j) \le K$
	18	$(i, j+1, p, 2)$	$(j+1-p)\mu_1$	$j+1 \le S$ and $(i+j+1) \le K$
			$(S-p)\mu_1 + p\mu_2$	$S < j+1 \le K$ and $(i+j+1) \le K$
	19	$(i, j+1, p+1, 2)$	$(p+1)\mu_2$	$j+1 \le S$ and $(i+j+1) \le K$
	20	$(i, j, p, 0)$	$\beta/2$	
Outgoing states	1	$(i-1, j, p, 2)$	$(i-p)\mu_1$	$i \le S$
			$(S-p)\mu_1 + p\mu_2$	$i > S$
	2	$(i, j-1, p, 2)$	$(j-p)\mu_1$	$j \le S$
			$(S-p)\mu_1 + p\mu_2$	$j > S$
	3	$(i-1, j, p-1, 2)$	$p\mu_2$	$0 < i \le S$ and $p > 0$
	4	$(i, j-1, p-1, 2)$	$p\mu_2$	$0 < j \le S$ and $p > 0$
	7	$(i, j+1, p, 2)$	λ	$(j < S$ and $S-(i+j-p) > 0$ and $(i+j+1) \le K)$ or $(S \le j < K$ and $(i+j+1) \le K)$
	8	$(i, j+1, p+1, 2)$	λ	$(S-(i+j-p) \le 0$ and $j < S$ and $(i+j+1) \le K)$
	9	$(i, j, p, 0)$	α	

3.4.3 Equilibrium states

Let $P(i, j, p, k)$ be the probability that the main memory system is in state (i, j, p, k). Let V be the set of states (i, j, p, k), where $i \in X$, $j \in X$, and $p \in Y(i, j)$. In the equilibrium state, the total incoming flows to state (i, j, p, k) are equal to the total outgoing flows from state (i, j, p, k). The equilibrium equations for $(i, j, p, k) \in V$ are given by,

$$\begin{aligned}&(r_1 + r_2 + r_3 + r_4 + r_{10} + r_{11})P(i, j, p, 0)\\&= r_{16}P(i + 1, j, p, 0) + r_{17}P(i + 1, j, p + 1, 0) + r_{18}P(i, j + 1, p, 0)\\&\quad + r_{19}P(i, j + 1, p + 1, 0) + r_{21}P(i, j, p, 1) + r_{22}P(i, j, p, 2),\end{aligned} \tag{3.14a}$$

$$\begin{aligned}&(r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + r_9)P(i, j, p, 1)\\&= r_{12}P(i - 1, j, p, 1) + r_{13}P(i - 1, j, p - 1, 1) + r_{16}P(i + 1, j, p, 1)\\&\quad + r_{17}P(i + 1, j, p + 1, 1) + r_{18}P(i, j + 1, p, 1) + r_{19}P(i, j + 1, p + 1, 1)\\&\quad + r_{20}P(i, j, p, 0),\end{aligned} \tag{3.14b}$$

$$\begin{aligned}&(r_1 + r_2 + r_3 + r_4 + r_7 + r_8 + r_9)P(i, j, p, 2)\\&= r_{14}P(i, j - 1, p, 2) + r_{15}P(i, j - 1, p - 1, 2) + r_{16}P(i + 1, j, p, 2)\\&\quad + r_{17}P(i + 1, j, p + 1, 2) + r_{18}P(i, j + 1, p, 2) + r_{19}P(i, j + 1, p + 1, 2)\\&\quad + r_{20}P(i, j, p, 0),\end{aligned} \tag{3.14c}$$

where r_c, $c \in [1, 22]$, equals the transfer rate of case c if the conditions of case c are satisfied and 0 otherwise.

The condition that the sum of all state probabilities equals one is given by,

$$\sum_{(i,j,p,k)\in V} P(i, j, p, k) = 1. \tag{3.15}$$

By considering the symmetric feature of states $(i, j, p, 1)$ and $(j, i, p, 2)$,

$$P(i, j, p, 1) = P(j, i, p, 2) \tag{3.16}$$

is satisfied. In (3.14a) and (3.15), $P(j, i, p, 2)$ is substituted by $P(i, j, p, 1)$ with (3.16). Then, (3.14c) can be omitted. The number of decision variables to be solved is reduced from 3Γ to 2Γ.

3.4.4 Blocking probability and average waiting time

Blocking probability $P_{\mathrm{b}}^{\mathrm{B}}$, which is the probability that a request incoming to the main memory system is blocked with $i + j = K$, or the request is not able to enter the queue, under the condition of ON state, is given by the following conditional probability.

$$\begin{aligned} P_{\mathrm{b}}^{\mathrm{B}} &= \sum_{i \in X} \sum_{p \in Y(i,j)} \sum_{k=1}^{2} P(i, K-i, p, k) \Bigg/ \frac{\frac{1}{\alpha}}{\frac{1}{\alpha} + \frac{1}{\beta}} \\ &= \frac{\alpha + \beta}{\beta} \sum_{i \in X} \sum_{p \in Y(i,j)} \sum_{k=1}^{2} P(i, K-i, p, k), \end{aligned} \tag{3.17}$$

note that $\frac{\frac{1}{\alpha}}{\frac{1}{\alpha}+\frac{1}{\beta}} = \frac{\beta}{\alpha+\beta}$ is the probability of ON state.

The average waiting time at the main memory system, W^{B} is the average duration time from when a request enters the main memory system until the request exits the main memory system. The average number of requests in the main memory system, L^{B}, is given by

$$\begin{aligned} L^{\mathrm{B}} &= \sum_{i \in X} \sum_{j \in X} \sum_{p \in Y(i,j)} \sum_{k=0}^{2} iP(i, j, p, k) + \sum_{i \in X} \sum_{j \in X} \sum_{p \in Y(i,j)} \sum_{k=0}^{2} jP(i, j, p, k) \\ &= 2 \sum_{i \in X} \sum_{j \in X} \sum_{p \in Y(i,j)} \sum_{k=0}^{2} iP(i, j, p, k). \end{aligned} \tag{3.18}$$

The right hand side for the second equality is derived by using

$$\sum_{i \in X} \sum_{j \in X} \sum_{p \in Y(i,j)} \sum_{k=0}^{2} iP(i, j, p, k) = \sum_{i \in X} \sum_{j \in X} \sum_{p \in Y(i,j)} \sum_{k=0}^{2} jP(i, j, p, k). \tag{3.19}$$

By using Little's formula [149], W^{B} is given by,

$$W^{\mathrm{B}} = \frac{L^{\mathrm{B}}}{\lambda'}. \tag{3.20}$$

λ' is the average arrival rate over both ON and OFF states in the main memory system. λ' is given by

$$\lambda' = \frac{\frac{\lambda}{\alpha}}{\frac{1}{\alpha} + \frac{1}{\beta}} = \frac{\lambda\beta}{\alpha + \beta}. \tag{3.21}$$

As with the analysis of random arrival of requests, let $\lambda_{\mathrm{e}}^{\mathrm{B}}$ be the throughput and $W_{\mathrm{e}}^{\mathrm{B}}$ be the average effective waiting time, which are defined by

$$\lambda_{\mathrm{e}}^{\mathrm{B}} = \lambda'(1 - P_{\mathrm{b}}^{\mathrm{B}}) \tag{3.22}$$

and

$$W_{\mathrm{e}}^{\mathrm{B}} = \frac{L^{\mathrm{B}}}{\lambda_{\mathrm{e}}^{\mathrm{B}}}, \tag{3.23}$$

respectively.

3.5 Evaluation

Based on the analytical results calculated with the model and its analyses shown in Sections 3.3 and 3.4, this work observes performance dependency on each system parameter and arrival pattern of requests of the proposed architecture.

3.5.1 Numerical results for random arrival of memory requests

This work evaluates $P_{\mathrm{b}}^{\mathrm{R}}$, $\lambda_{\mathrm{e}}^{\mathrm{R}}$ and $W_{\mathrm{e}}^{\mathrm{R}}$ of the proposed architecture and investigate their dependency on ρ^{R} and μ_2, by using the analysis presented in Section 3.3. The M/M/S/K model is used as a reference model. This work sets $K = 100$, and the arrival rate of λ is the same for both models. This work sets $S = 32$ for both models unless otherwise stated. In M/M/S/K, the service rate is $\mu = 1$, and, in the proposed architecture, $\mu_1 = \mu = 1$. Let ρ^{R} be the traffic load, which is defined by,

$$\rho^{\mathrm{R}} = \frac{\lambda}{S\mu}. \tag{3.24}$$

The analytical results are obtained by using a computer with 3.60GHz Intel Core i7-7700 CPU and 32GB memory. In the case of $S = 32$ and $K = 100$, the average computation time to obtain $P_{\mathrm{b}}^{\mathrm{R}}$ for each set of μ_2 and ρ^{R} in the proposed architecture is 252 [sec], where the number of states in the analysis is 10607.

Figure 3.8 shows the blocking probability dependency on ρ^{R} with different μ_2. In the proposed architecture, the blocking probability increases with ρ^{R}, and decreases as μ_2 increases. The blocking probability of the proposed architecture with $\mu_2 = 0.5$ is close to, but slightly higher than, that of M/M/S/K.

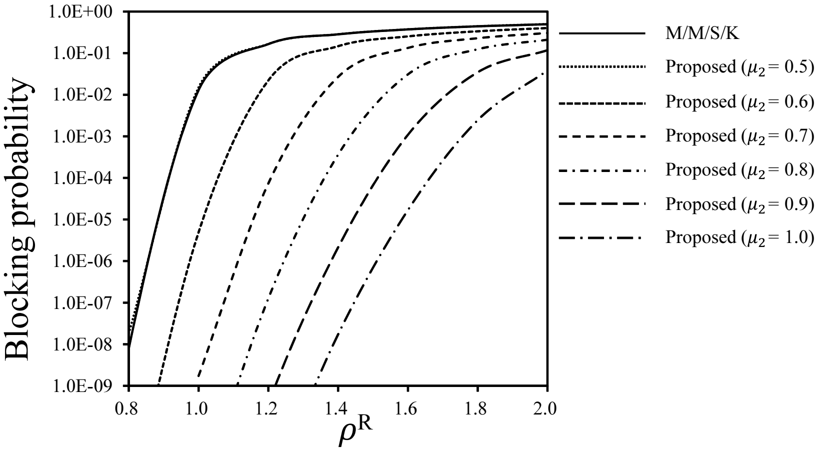


Figure 3.8: Blocking probability depending on ρ^{R} with different μ_2.

This is explained by the observation that, when two main memory systems have the same value of the product of the number of servers and the service rate, the one with larger service rate outperforms the other. In addition, 32 servers with service rate $\mu = 1$, all requests queued in the main memory system are served, outperform 64 ($= 32 \times 2$) servers with service rate $\mu_2 = 0.5$, half of which serve requests queued in one of the two separate queues; the former has greater statistical multiplexing effect than the latter.

Figure 3.9 shows the blocking probability dependency on μ_2 with $\rho^{\mathrm{R}} = 1.2$. In the proposed architecture, the blocking probability decreases as μ_2 increases. The blocking probability of the proposed architecture with $\mu_2 = 1$ is close to, but slightly higher than, that of M/M/S/K with $S = 64$. This is explained by comparing 64 servers with service rate $\mu = 1$ and 64 ($= 32 \times 2$) servers with service rate $\mu_2 = 1$ as with the observation on Figure 3.8.

Figure 3.10 show the throughput dependency on ρ^{R} with different μ_2. The throughput in M/M/S/K increases with $\rho^{\mathrm{R}} \leq 1$, and is saturated with $\rho^{\mathrm{R}} > 1$.

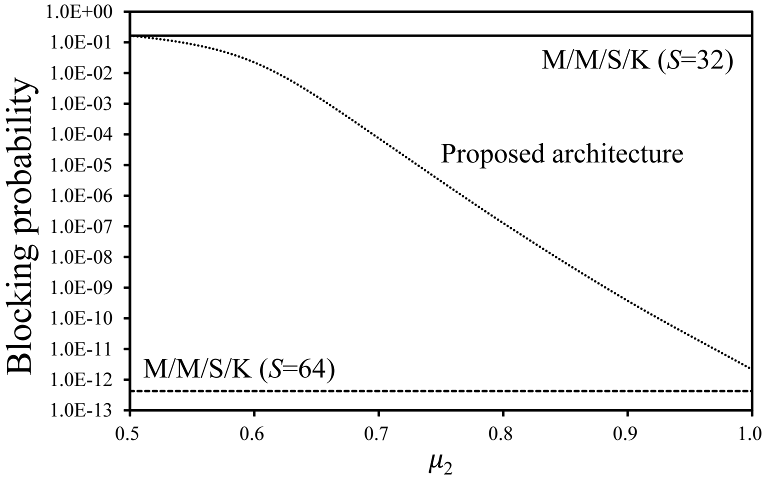


Figure 3.9: Blocking probability depending on μ_2 with $\rho^{\mathrm{R}} = 1.2$.

On the other hand, the throughput of the proposed architecture saturates at a larger point than M/M/S/K. The saturated throughput increases with μ_2.

Figure 3.11 show the effective waiting time dependency on ρ^{R} with different μ_2, where the effect of blocked requests is eliminated. The effective waiting time decreases as μ_2 increases. Figure 3.12 shows the effective waiting time dependency on μ_2. Similar dependency to that on Figure 3.9 are observed.

3.5.2 Numerical results for bursty arrival of memory requests

This work evaluates $P_{\mathrm{b}}^{\mathrm{B}}$, $\lambda_{\mathrm{e}}^{\mathrm{B}}$ and $W_{\mathrm{e}}^{\mathrm{B}}$ of the proposed architecture for IPP and investigate their dependency on ρ^{B} and μ_2, by using the analysis presented in section 3.4. This work compares the proposed architecture for IPP with a Poisson arrival process. This work sets $K = 100$ and $S = 32$ unless otherwise stated. Let ρ be the traffic load, which is defined by,

$$\rho^{\mathrm{B}} = \frac{\lambda'}{S\mu} = \frac{\lambda\beta}{(\alpha+\beta)S\mu}. \tag{3.25}$$

In the second equality, (3.21) is used.

For performance comparison, this work uses the same ρ^{B} for different models. The analytical results are obtained by using a computer with 3.60GHz Intel

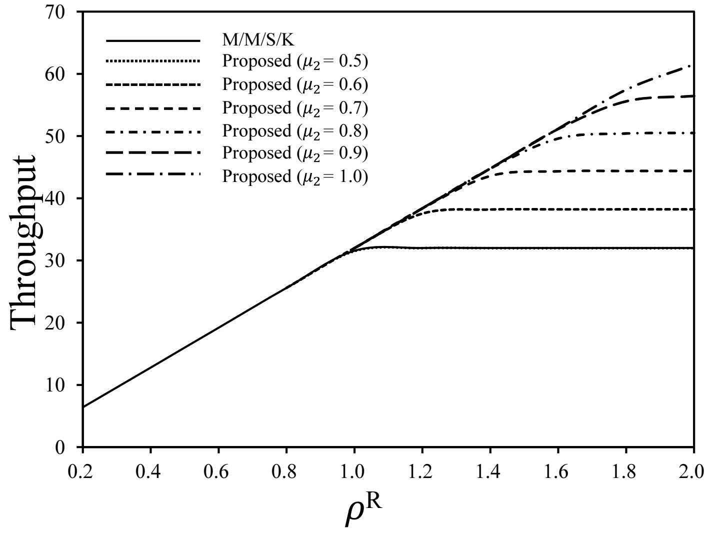


Figure 3.10: Throughput depending on ρ^{R} with different μ_2.

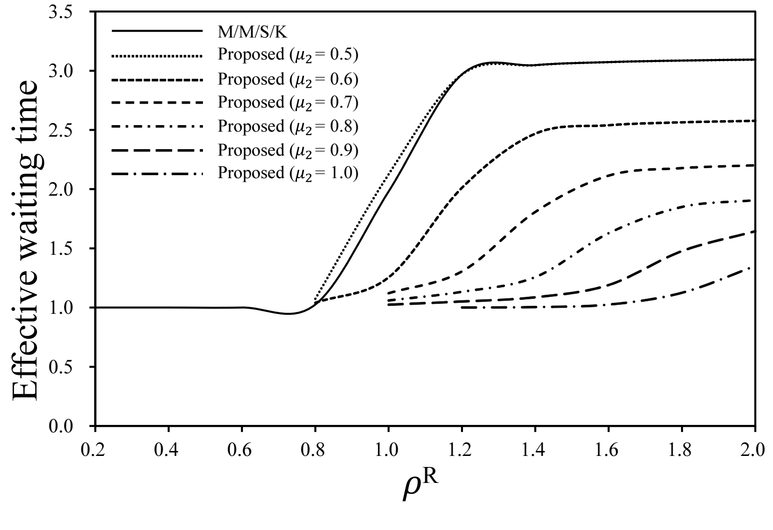


Figure 3.11: Effective waiting time depending on ρ^{R} with different μ_2.

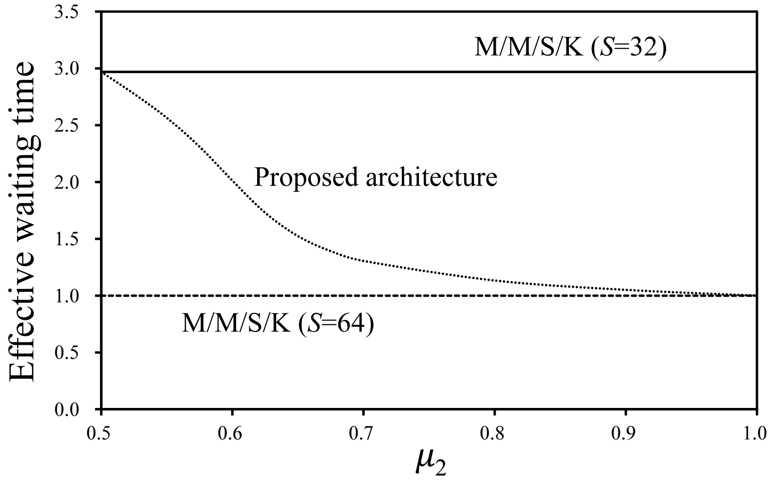


Figure 3.12: Effective waiting time depending on μ_2 with $\rho^{\mathrm{R}} = 1.2$.

Core i7-7700 CPU and 32GB memory. In the case of $S = 32$ and $K = 100$, the average computation time to obtain P_{b} for each set of μ_2 and ρ^{B} in the proposed architecture is 853 [sec], where the number of states in the analysis is 21214.

This work introduces parameters, $h > 0$ and $l > 0$, which are defined by $l = \frac{\alpha}{\lambda}$ and $h = \frac{\alpha}{\beta}$. Then, $\lambda' = \frac{\alpha}{(h+1)l}$ and $\rho^{\mathrm{B}} = \frac{\alpha}{(h+1)lS\mu}$. When $h \to 0$, IPP approaches a Poisson arrival process. Note that, with $h \to 0$, each packet continues to have the destination of the same bank, which is different from the model presented in Section 3.3. For any h, when $l \to \infty$, IPP approaches a Poisson arrival process and the proposed architecture with IPP approaches the model presented in Section 3.3, where the destination of each packet is randomly assigned to either bank.

Figure 3.13 shows the blocking probability dependency on l and h with $\rho^{\mathrm{B}} = 1.2$ for $\mu_2 = 0.7$. As h becomes large, the blocking probability increases. As l becomes large, the blocking probability decreases. This work observes that, when $l \to \infty$, the blocking probability of IPP approaches that of the Poisson arrival process presented in Section 3.3 for any h. The lower h is, the faster the blocking probability of IPP approaches that of the Poisson arrival process. $h \to 0$ indicates that ON state probability is close to 1, and $l \to \infty$ indicates that ON state period is close to zero. Each situation is equivalent to the Poisson arrival process, which is a special case of IPP.

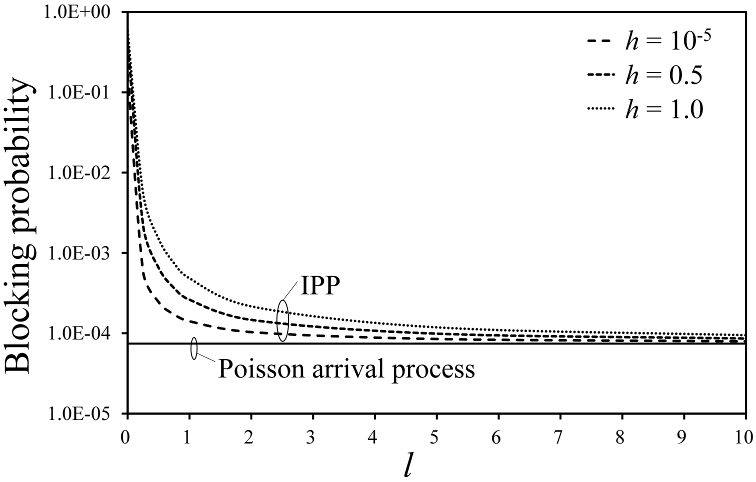


Figure 3.13: Blocking probability depending on l and h with ρ^{B} = 1.2 and μ_2 = 0.7.

Figure 3.14 shows the blocking probability dependency on ρ^{B} and h with l = 1.0 for μ_2 = 0.7.

Figures 3.15 and 3.16 show the throughput and effective waiting time dependency on l and h with ρ^{B} = 1.2 for μ_2 = 0.7, respectively.

3.5.3 Packet Processing Performance

Packet processing performance of the proposed architecture can be calculated by using the numerical results shown in Sections 3.5.1 and 3.5.2, assuming that packet processing performance in network virtualization based on the general-purpose computers is bounded by memory performance. In the following calculation of packet processing performance, this work focuses on IP routing as an example packet processing application.

This work considers DIR-24-8-BASIC [13] for an example IP address lookup algorithm, in which packet processing of IP routing is finished within one or two memory accesses at most. Its lookup table has, 2^{32} entries, in nature, which correspond to the whole IPv4 address space. The content in each entry is the next hop information corresponding to the prefix of the entry. In detail,

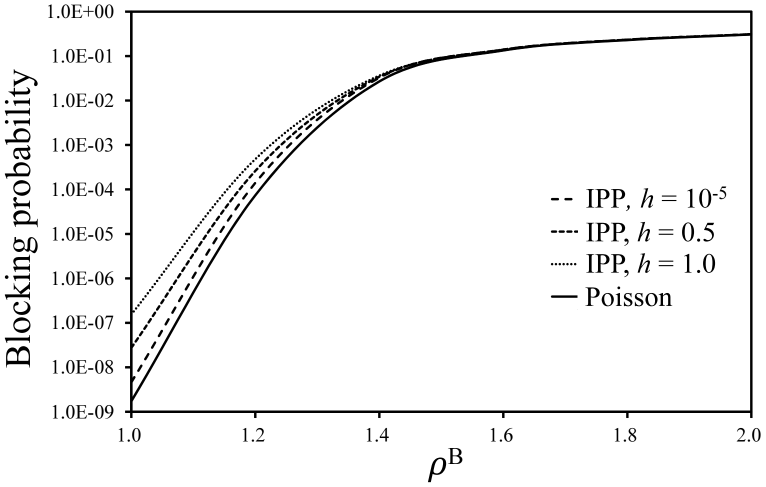


Figure 3.14: Blocking probability depending on ρ^{B} and h with l = 1.0 and μ_2 = 0.7.

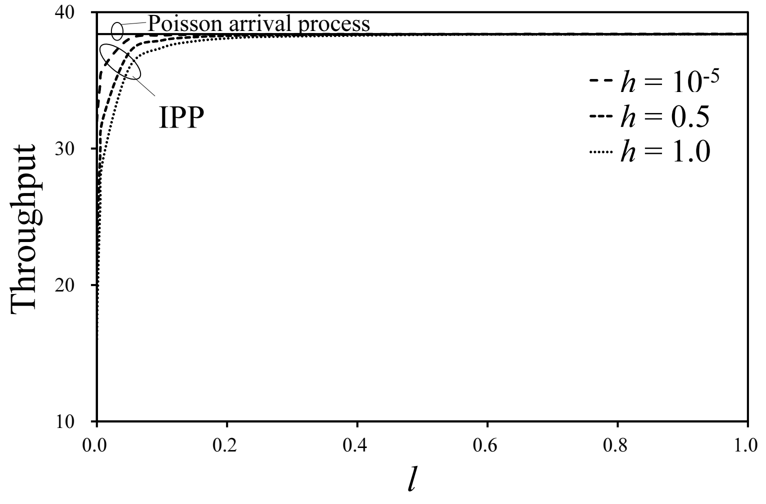


Figure 3.15: Throughput dependency on l and h with ρ^{B} = 1.2 and μ_2 = 0.7.

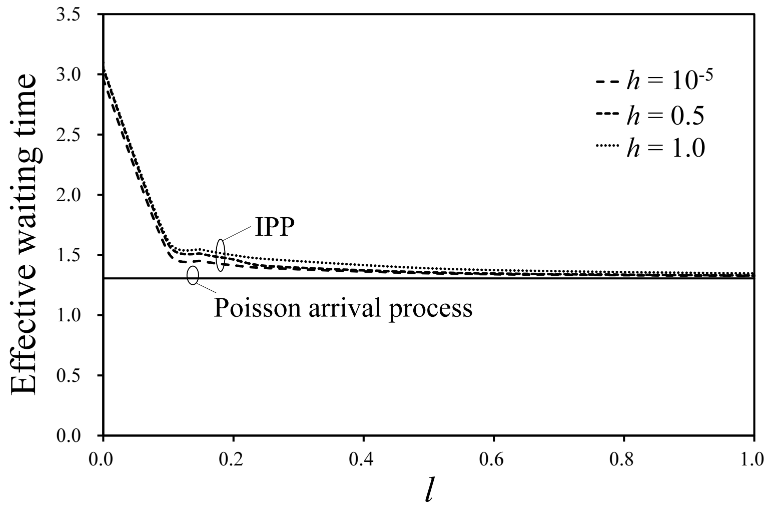


Figure 3.16: Effective waiting time dependency on l and h with $\rho^{\mathrm{B}} = 1.2$ and $\mu_2 = 0.7$.

based on typical traffic patterns, the lookup tables comprise two lookup tables called TBL24 and TBLlong, each of which has the entries corresponding to the upper 24 bits and lower 8 bits, respectively. In the algorithm, TBL24 is firstly accessed, and if the table lookup result of TBL24 is the next hop information, the LPM is finished. If a pointer to TBLlong is returned as a result of searching TBL24, TBLlong is also accessed to obtain the next hop information. Thereby, the LPM is finished within two memory accesses.

Consequently, the packet processing performance of IP routing can be calculated using the number of acceptable memory accesses per unit of time as obtained from the numerical results in Sections 3.5.1 and 3.5.2 and the number of necessary memory accesses to the IP address lookup per packet, and the additional delay of the distributor based on a hash function. Packet processing latency is obtained as the number of required memory accesses times the sum of the average effective waiting time and the additional delay of the distributor.

This work assumes that the hash function works as a pipeline, where it does not affect any other performance metric of the proposed architecture except for the latency.

For example, let service rate μ be 8 M services per second, which is an estimate since typical latency inside the HMC itself is usually taken to be between 100-180 ns with an average of 125 ns [156,157], and let the traffic load be 1.2. In this example, this work assumes the service rate of interleaved bank $\mu_2 = 0.7\mu$, $l = 0.1$, $h = 0.5$, and the additional delay of the distributor is 10 ns, which corresponds to one clock cycle at 100 MHz circuits in the FPGA. This work also assumes that each IP address lookup requires two memory accesses, each of which is associated with a request distribution based on a hash function. Table 3.5 lists the calculation results for proposed architecture with random arrival of requests that with bursty arrival of requests, and M/M/S/K for reference, respectively.

Table 3.5: Example of packet processing performance.

Model	Throughput (Gbps) at 64B packet	Latency (ns)
Proposed (Poisson)	79	471
Proposed (IPP)	78	542
M/M/S/K	66	887

3.6 Direction to expansion of analytical model for general $N \geq 2$

This work considers the analytical model for a system with general $N \geq 2$. A state in the system is expressed by a vector, that consists of the following components. First, $i_n \in [0, K]$ is the number of requests for bank $n \in [1, N]$. Second, $p_{i_n i_{n'}} \in [0, S]$, where $n, n' \in [1, N]$ and $n \neq n'$, is the number of requests being served with 2-interleaving for banks n and n'. The number of $p_{i_n i_{n'}}$ is ${}_N C_2$, where ${}_n C_m = \frac{n!}{(n-m)!m!}$. Third, $p_{nn'n''} \in [0, S]$, where $n, n', n'' \in [1, N]$, $n \neq n'$, $n' \neq n''$, and $n'' \neq n$, is the number of requests being served with 3-interleaving for banks n, n', and n''. The number of $p_{nn'n''}$ is ${}_N C_3$. In the same way, $p_{nn'n''n'''\dots}$ is the number of requests being served with $(N-1)$-interleaving. The number of $p_{nn'n''n'''\dots}$ for $(N-1)$-interleaving is ${}_N C_{N-1} = N - 1$. Finally,

$p_{123\cdots N} \in [0, S]$ is the number of being served requests with N-interleaving for all banks.

3.7 Chapter summary

This chapter proposed a parallel memory system architecture that uses a 3D-stacked memory to increase the memory parallelism of main memory in general-purpose computers for packet processing in network virtualization. A database searched in packet processing is divided into several partial databases, each of which is deployed in memory banks in a memory channel, and the database in the memory channel is copied to the other memory channels in the 3D-stacked DRAM so that the proposed architecture enhances the memory parallelism by leveraging the memory channel-level parallelism and bank interleaving. Incoming memory requests are distributed based on a hash function, according to the packet content, into an appropriate memory channel-bank set that contains a partial database in the 3D-stacked DRAM. This work introduced an analytical model of the proposed main memory system architecture that considers the bank interleaving, for two traffic patterns where the arrivals of memory requests are either random or bursty. The analytical results for random arrival of memory requests observed the performance dependency of the proposed architecture on traffic load and the number of bank interleaving. The analytical results for bursty arrival of memory requests observed the performance dependency of the proposed architecture on input traffic burstiness. Based on these analytical results, the calculated packet processing performance, for which IP routing is taken as an example, showed that the proposed architecture increases packet processing performance up to 80 Gbps for the smallest-sized IP packet involving both arrival patterns of random and bursty memory requests.

Chapter 4

Parallel memory system architecture using interleaved 3D-stacked memory and private cache memory

This chapter proposes a parallel memory system architecture that uses a 3D-stacked memory and on-chip private cache memories to reduce memory access latency in packet processing in network virtualization. A part of the work in this chapter was presented in [158]. Modern multi-core CPUs in general-purpose computers have several levels of on-chip cache memories, some of which are private to each CPU core, and the other is shared among all the CPU cores. This work explores the memory system architectures that integrate the on-chip cache memories with the off-chip interleaved 3D-stacked DRAM-based main memory. The proposed architecture integrates the private cache memories of each CPU core and the off-chip 3D-stacked DRAM. Frequently accessed entries of the packet processing database stored in the 3D-stacked DRAM are cached in the on-chip cache memories.

The proposed architecture is compared to the following two reference memory system architectures: one with the on-chip private cache memories, the on-chip shared LLC, and the off-chip 3D-stacked DRAM and one without on-chip cache memory and with the off-chip 3D-stacked DRAM. In this chapter, these reference architectures are hereinafter referred to as the architecture with LLC and the architecture without any cache, respectively. As a result of the queueing model-based simulations, this work observes that the proposed memory system architecture reduces memory access latency by 58 % and 1.8 % and increases throughput by 104 % and 1 % with reducing the blocking probability by 91 % and 18 %, compared to the reference architecture with LLC and the architecture without any cache, respectively. The simulation results, in which the proposed architecture with the on-chip private cache memories outperforms the reference architecture with LLC, also suggest that memory parallelism in the on-chip cache memories is inevitable for packet processing in network virtualization when the main memory parallelism exists.

The rest of this chapter is organized as follows. Section 4.1 presents the proposed architecture. Section 4.2 describes the system models of the proposed architecture and the reference architectures. Section 4.3 presents performance evaluation, in which the proposed architecture is compared with the reference architectures. Section 4.4 summarizes this chapter.

4.1 Proposed architecture

Figure 4.1 shows an overview of the proposed memory system architecture. It consists of a multi-core CPU that includes on-chip two levels of cache memories dedicated to each CPU core, an FPGA, a 3D-stacked DRAM, a DRAM, and network interfaces.

The packet processing flow in the proposed architecture is similar to that in Section 3.1. However, the proposed architecture has on-chip cache memories in addition to the architecture in 3.1. Therefore, each incoming packet is processed as follows.

(Step 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. The assigned CPU core reads the header information of the packet from the packet buffer in the DRAM.

(Step 2) The CPU core issues memory requests to read the database stored in the on-chip cache memories and the 3D-stacked DRAM in order to decide

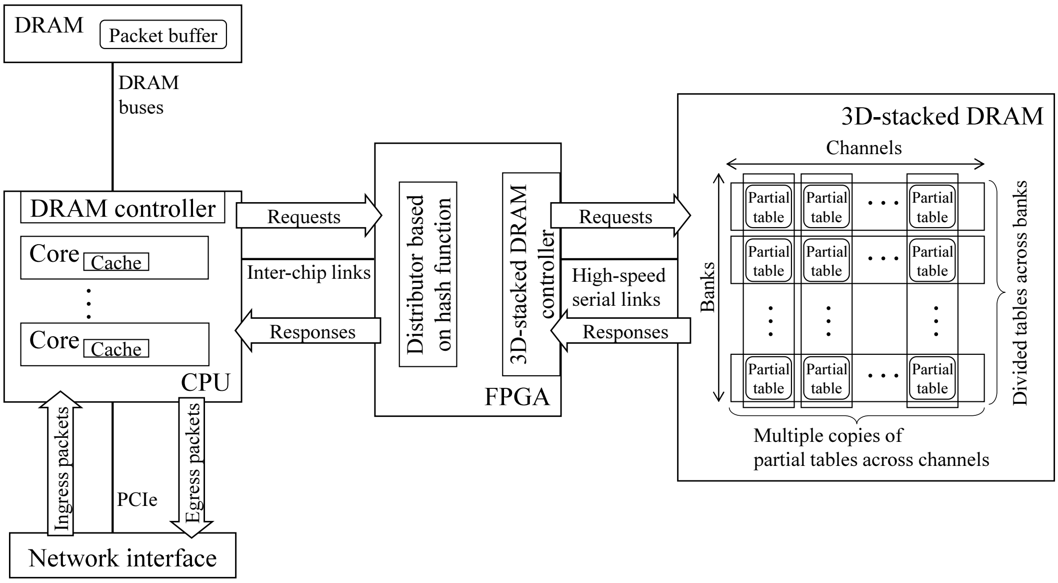


Figure 4.1: Overview of proposed architecture.

the next action for the packet. When searching the database, the on-chip cache memories is firstly accessed. If the required entry is not found in the on-chip cache memories, the 3D-stacked DRAM is accessed, and the cache line that includes the found entry is cached based on a particular cache replacement policy.

(Step 3) After finishing the lookup and determining the next action, the CPU core sends the packet including its payload from the packet buffer in the DRAM through the network interface.

This work focuses on the packet processing step 2 in the above description, which requires high memory performance, compared to the steps 1 and 3. As described in Section 2.1, performance of the steps 1 and 3 can be improved by using the packet I/O acceleration methods such as DPDK. This work considers packet classification and database searching for table lookup tasks. Additionally, the database, such as lookup table data, is separated into several partial databases as many as the number of banks in a memory channel. Each database in a memory channel is copied to the other memory channels in the 3D-stacked DRAM, as described in Section 3.1. In the proposed architecture, since the on-chip LLC is not integrated in the CPU, the 3D-stacked memory-based main memory behaves like an off-chip LLC that has a larger capacity and memory parallelism than on-chip shared LLC.

An FPGA is used for the same purpose as presented in Section 3.1, where it is used to connect the CPU with the 3D-stacked DRAM via inter-chip connections such as Intel QPI or UPI [141–143, 159]. In the FPGA, as with in Section 3.1, the logic circuit for the hash-function-based memory requests distributor and the memory controller of the 3D-stacked DRAM is deployed. The hash-function-based distributor distributes memory requests into an appropriate memory channel-bank sets in the 3D-stacked DRAM. The memory requests that enter the FPGA are the memory requests that miss the on-chip L2 cache memory and are destined to the main memory that is the 3D-stacked DRAM. The mechanism and behavior of cache memories are explained in Sections 1.7.2 and 4.2. The memory controller of the 3D-stacked DRAM interfaces the 3D-stacked DRAM.

4.2 System model

This work considers the three system models shown in Figures 4.2, 4.3, and 4.4, where each system model corresponds to the proposed architecture and two reference architectures. Every architecture has the off-chip 3D-stacked DRAM that is connected to the CPU. The behavior of each architecture is described in the subsequent subsections. Since the database placement and memory request distribution are explained in detail in Section 3.1, the description of each system model is focused on the on-chip cache memory systems in the CPU. Additionally, in this work, the on-chip cache memories in the CPU store the copies of frequently used data in the 3D-stacked DRAM based on the LRU policy.

4.2.1 System model of proposed architecture

Figure 4.2 shows the system model of the proposed architecture that integrates the on-chip two levels of private cache memories of each CPU core with the off-chip 3D-stacked DRAM. The on-chip private cache memories are the L1 cache memory and L2 cache memory, each of which is dedicated to the corresponding

CPU core.

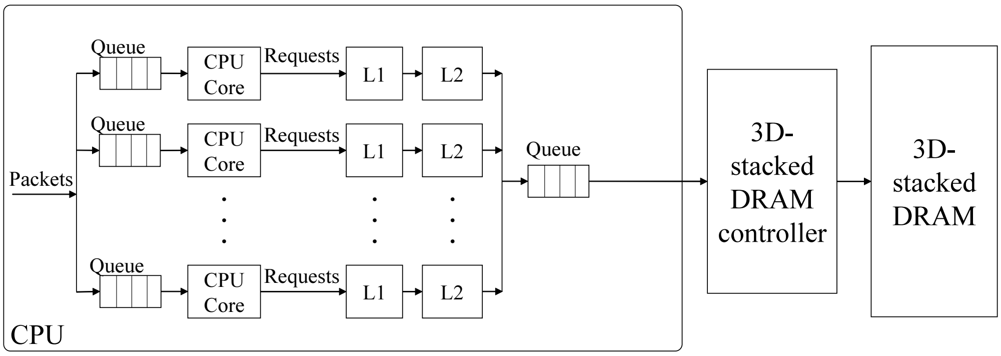


Figure 4.2: System model of proposed architecture.

When a packet enters the system, it is randomly assigned to one of the CPU cores regardless of the CPU core state. After its assignment to the CPU core, the packet is enqueued into the corresponding queue in front of the assigned CPU core and is processed following the FCFS policy. The total number of requests, which includes waiting requests for all the queues and the requests being processed by the CPU cores, is limited in the system. The processed request firstly accesses the private cache memories of the CPU core in an increasing order of cache levels, where the L1 cache memory is firstly accessed. If the corresponding table entry of the request is not found, which is called a miss, in any level of cache memory, the request accesses the next level of cache memory if there is; otherwise, the request is transferred to a queue to wait for being distributed to an appropriate memory channel-bank set of the 3D-stacked DRAM.

The main memory system model for table lookup is explained in Chapter 3. Note that at any time, only one request is processed by each CPU core; the processing of a request is completed when its corresponding table entry is found, which is called a hit in any part of the system. All the requests from different CPU cores access the 3D-stacked DRAM according to the FCFS policy.

After the hit, the copy of the cache line that includes the corresponding table entry of the request is stored in the L1 cache memory of the corresponding CPU core as the most recently used content. For any level cache memory of the corresponding CPU core, if it is a miss, the LRU content in the cache memory is evicted to the next level cache memory. For example, if the corresponding table entry of the request is found in the 3D-stacked DRAM, the cache line including it is copied to the L1 cache memory of the corresponding CPU core, and the LRU content in the L1 cache memory is evicted to the L2 cache memory. Meanwhile, the LRU content in the L2 cache memory is dropped.

Let C_{sys} denote the number of CPU cores in the system. Let N_{ent} represent the total number of entries in the system. The maximum number of requests in the system is represented by K. A request incoming to the system is blocked if the number of accommodated requests in the system is K. The size of each queue associated with each CPU core is at least K, which means that no loss of request occurs in each queue. The size of queue associated with the 3D-stacked DRAM controller is considered to be equal with the number of CPU cores, C_{sys}. The memory capacities of L1 cache memory, L2 cache memory, and 3D-stacked DRAM are considered as M_{L1}, M_{L2}, M_{3D}, respectively. Let B denote the size of a cache line. The size of each table entry is represented by b. B/b table entries are copied to the L1 cache memory when there is a miss.

4.2.2 System models of reference architectures

Figure 4.3 shows the system model of the reference architecture, where the off-chip 3D-stacked DRAM is combined with the multi-core CPU with L1 and L2 cache memories of each CPU core and on-chip shared LLC. Figure 4.4 is the system model of the reference architecture, where the off-chip 3D-stacked DRAM is combined with the multi-core CPU without any on-chip cache memories.

The behavior of these models follows that of the proposed architecture; the differences are as follows. In the system model of the reference architecture that has the on-chip shared LLC, a request that misses the L2 cache memory is transferred to a queue in front of the on-chip shared LLC to wait for the access to the on-chip shared LLC. If it is a request that misses the shared LLC, the request is transferred to the 3D-stacked DRAM. When the LRU content in the L2 cache memory is evicted to the on-chip shared LLC, the LRU content in the on-chip shared LLC is dropped.

In the system model of the reference architecture without any on-chip cache memories, every request directly transferred to a queue to wait for the access to the 3D-stacked DRAM. All the requests from different CPU cores access the on-chip shared LLC or the 3D-stacked DRAM according to the FCFS policy. The memory capacity of the on-chip shared LLC is denoted as M_{LLC}.

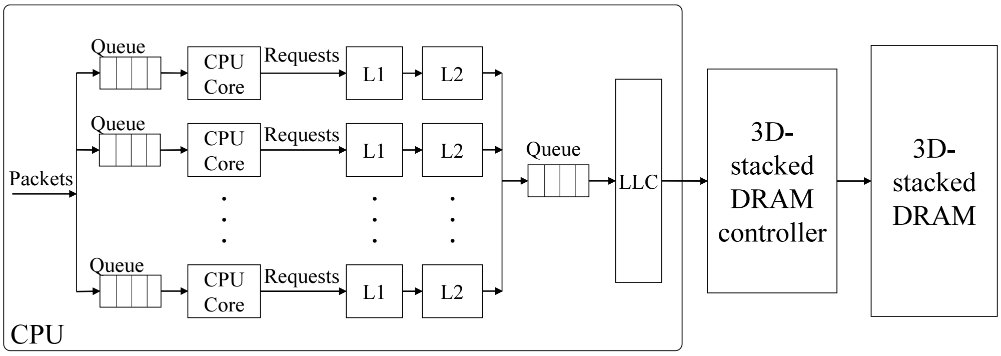


Figure 4.3: System model of reference architecture with LLC.

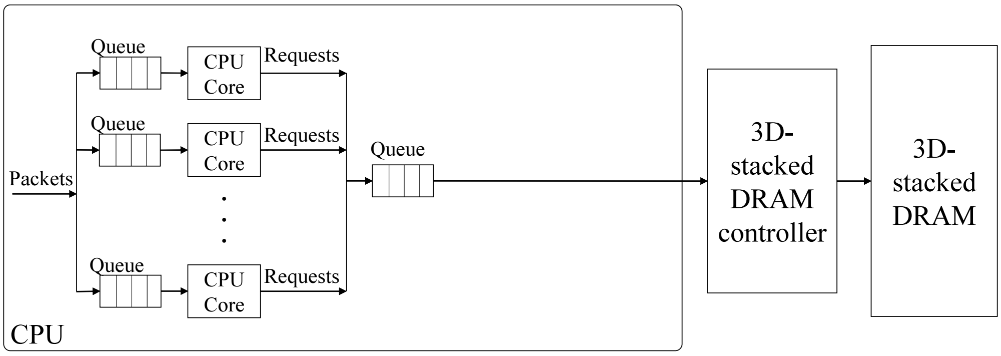


Figure 4.4: System model of reference architecture without any cache.

4.3 Evaluation

4.3.1 Traffic model

To evaluate the effectiveness of caching, some studies use the actual traffic traces, such as the works in [160–162], and others generate the synthetic traces with considering the content popularity, such as the works in [163, 164], where the content in traffic, such as the Web page requests and the IP addresses lookup requests, is considered to follow a Zipf-like distribution [165]. It indicates a behavior of the traffic that a few most popular contents are requested in high probabilities, and a large proportion of contents are requested in low probabilities.

This work assumes a traffic model of memory requests, in which a request arrives at the memory system based on a Poisson arrival process with the average rate of λ. Let N_{ent} represent the total number of table entries stored in the system, which are sorted in decreasing order of popularity. This work assumes that the requested content follows a Zipf-like distribution, where the relative probability of requesting for the ith most popular table entry is expressed as $\frac{1}{i^{\alpha}}$, which leads to the probability with normalizing constant as $\frac{\frac{1}{i^{\alpha}}}{\sum_{i=1}^{N_{\text{ent}}} \frac{1}{i^{\alpha}}}$. Note that the Poisson and Zipf-like distributions in the traffic model of this work are orthogonal or independent from each other.

According to the work in [165], the value of α in the Zipf-like distribution varies for different traffic traces. It is reported that the special case of $\alpha = 1$, which is known as the strict Zipf's law, is not appropriate for the content distributions in traffics, such as the Web page requests and the IP addresses lookup requests. Typically, the value of α is in the range of $0 < \alpha < 1$. α in traces from a homogeneous environment appears to be larger than that in traces from a more diversified user population. In other words, as α increases, more requests are concentrated on a few most popular contents. This work sets the value of α as 0.83 [165].

This work assumes that the probability of accessing each memory bank of the 3D-stacked DRAM is equal when the content popularity is considered. This work adopts the model for the memory system architecture presented in Chapter 3, where requests are randomly assigned to memory banks in the 3D-stacked DRAM.

4.3.2 System assumption

This subsection introduces the assumptions of the system model for the simplicity of the numerical simulations.

This work assumes 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}.

This work assumes that the processing times of cache memories and 3D-stacked DRAM follow exponential distributions. This work considers 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 [166] automatically increases clock frequency of CPU cores for peak loads if the power consumption and temperature of the CPU are below the specification limits. Consequently, the processing times in cache memories are assumed to follow exponential distributions. The processing time of 3D-stacked DRAM includes the DRAM access latency due to DRAM specific behavior such as precharging the row buffer when accessing another row. This work assumes that the processing rate of the interleaved banks in the off-chip 3D-stacked DRAM is 0.7 times of that of without bank interleaving.

This work also assumes that the memory system is in the steady condition in which there is no memory write request such as updating the table, which allows the system models to ignore cache coherency. In addition, the CPU reads data from each level of cache memory or the 3D-stacked DRAM in 8-byte groups.

This work assumes that the total number of entries, N_{ent}, the maximum number of memory requests in the system, K, and the memory capacities of each level of cache memory, M_{L1}, M_{L2}, M_{LLC}, are smaller than those of today's general-purpose computers. These assumptions allow the model to finish the numerical simulations within a practical time. Without this assumption on the memory capacity of each cache memory and the number of table entries, the numerical simulations do not finish in a reasonable time. For instance, if simulation parameters are set as $M_{\mathrm{L1}} = 64$ [KiB], $M_{\mathrm{L2}} = 512$ [KiB], $M_{\mathrm{LLC}} = 28$ [MiB], the estimated time to obtain a one-plot result is at least one month by using the simulation environment that is described in Section 4.3.4.

4.3.3 Blocking probability and average waiting time

Let R represent a set of requests incoming to the system during a certain period time. $|R|$ denotes the total number of requests in R. The number of rejected requests that come to the system during the period is represented by $|R_{\mathrm{b}}|$, where R_{b} denotes the set of rejected requests. Blocking probability P_{b}, which is a probability that a request incoming to the system is rejected, is defined by $P_{\mathrm{b}} = \frac{|R_{\mathrm{b}}|}{|R|}$. Throughput, λ_{e}, is defined by $\lambda_{\mathrm{e}} = \lambda(1 - P_{\mathrm{b}})$. For accepted request $r \in R \backslash R_{\mathrm{b}}$, let t_r represent its waiting time to be processed by corresponding CPU core. Average effective waiting time, W_{e}, is defined by $W_{\mathrm{e}} = \frac{\sum_{r \in R \backslash R_{\mathrm{b}}} t_r}{|R| - |R_{\mathrm{b}}|}$.

In the numerical analysis, this work considers a 3D-stacked DRAM with two banks and S memory channels. The processing rates of a memory channel of the 3D-stacked DRAM with and without memory interleaving are denoted as μ_2 and μ_1, respectively. Let ρ be a traffic load, which is defined by $\rho = \frac{\lambda}{S\mu}$.

This work considers the IP address lookup based on DIR-24-8-BASIC [13] for the benchmark of the packet processing, where the size of each entry is 2 byte. Thus this work sets $b = 2$ [B]. This work considers that the multi-core CPU has $C_{\mathrm{sys}} = 28$ CPU cores, and the off-chip 3D-stacked DRAM has $S = 32$ memory channels. The memory capacity of the off-chip 3D-stacked DRAM is considered to be $M_{\mathrm{3D}} = 4$ [GiB], which is large enough to store all the entries in the memory system. This work sets the service rates of each level of cache memory and that of the off-chip 3D-stacked DRAM as $\mu_{\mathrm{L1}} = 100$, $\mu_{\mathrm{L2}} = 50$, $\mu_{\mathrm{LLC}} = 10$, $\mu = \mu_1 = 1$, according to the works in [156, 157, 167]. Based on the aforementioned descriptions and assumptions, this work sets $K = 100$, $\alpha = 0.83$, $M_{\mathrm{L1}} = 128$ [B], $M_{\mathrm{L2}} = 512$ [B], $M_{\mathrm{LLC}} = 4096$ [B], $N_{\mathrm{ent}} = 2 \times 10^4$, and $\mu_2 = 0.7$. This work uses $\rho = 0.7$ unless otherwise stated.

4.3.4 Numerical results

This work uses Intel Xeon Silver 4216 2.10 GHz 16-core CPU with 128 GB memory to run the simulations based on Python 3.7.

This work presents the numerical simulation results of three system models, each of which corresponds to the following architecture: (a) proposed architecture, (b) architecture with LLC, and (c) architecture without any cache. This subsection labels these three as *Proposed*, *With LLC*, and *Without any cache.*

Figure 4.5 shows the blocking probabilities for the proposed architecture, the architecture with LLC, and the architecture without any cache; the set of values of ρ is considered as $\{0.1, 0.2, 0.3, \cdots, 1.8, 1.9, 2.0\}$. This work observes that as the traffic load increases, the blocking probability increases for every architecture. The blocking probability of the architecture with LLC increases rapidly compared to those of the other two architectures. This indicates that the on-chip shared LLC becomes a bottleneck due to the concentration of memory requests from multiple CPU cores as the traffic load increases. When $\rho = 0.7$, the blocking probabilities for the proposed architecture and the architecture without any cache are less than 10^{-1}.

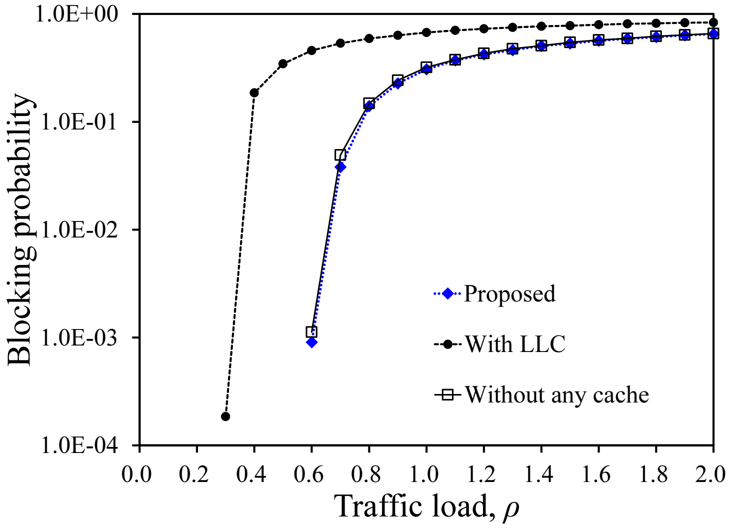


Figure 4.5: Blocking probability depending on traffic load for different architectures.

Figure 4.6 shows the blocking probabilities for the proposed architecture, the architecture with LLC, and the architecture without any cache; the set of M_{L1} is considered as $\{128, 192, 256, 384, 512\}$ [B]. This work observes that the proposed architecture outperforms the other two architectures. In addition, the blocking probabilities are almost independent of the capacity of the L1 cache memory, which does not need to increase the capacity of the expensive L1 cache memory.

Figures 4.7, 4.8, and 4.9 show the blocking probabilities, average effective waiting times, and throughputs for the proposed architecture, the architecture

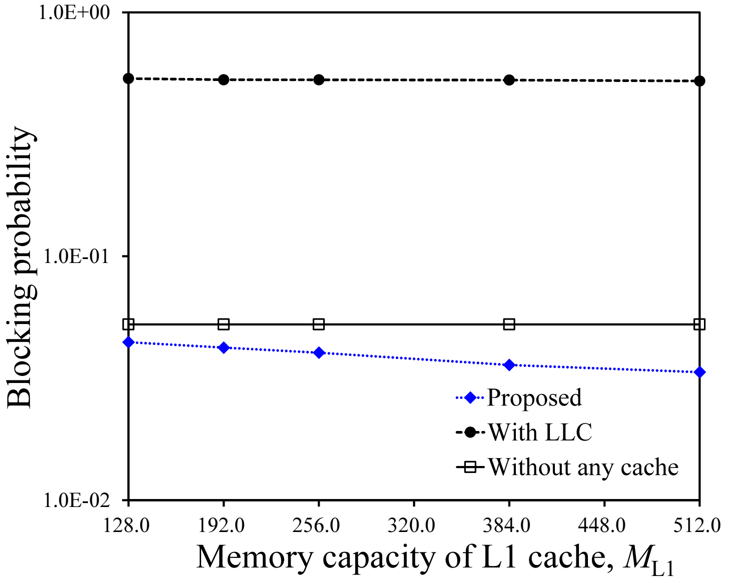


Figure 4.6: Blocking probability depending on memory capacity of L1 cache for different architectures.

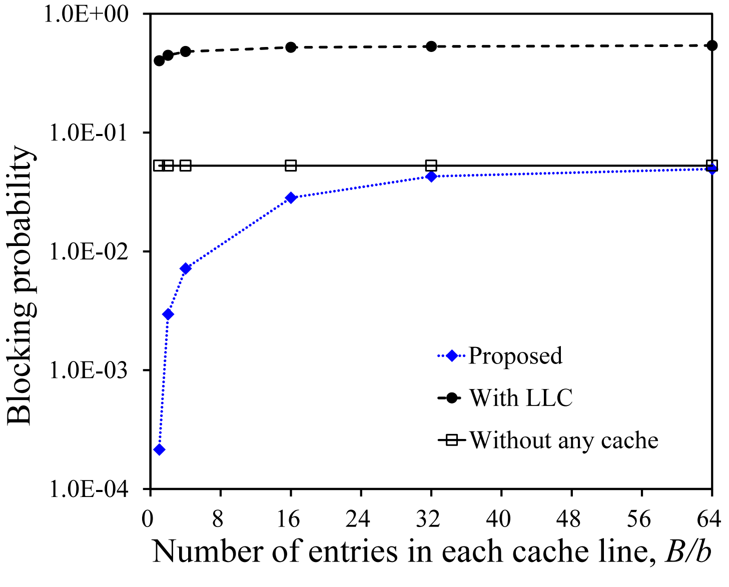


Figure 4.7: Blocking probability depending on number of entries in each cache line for different architectures.

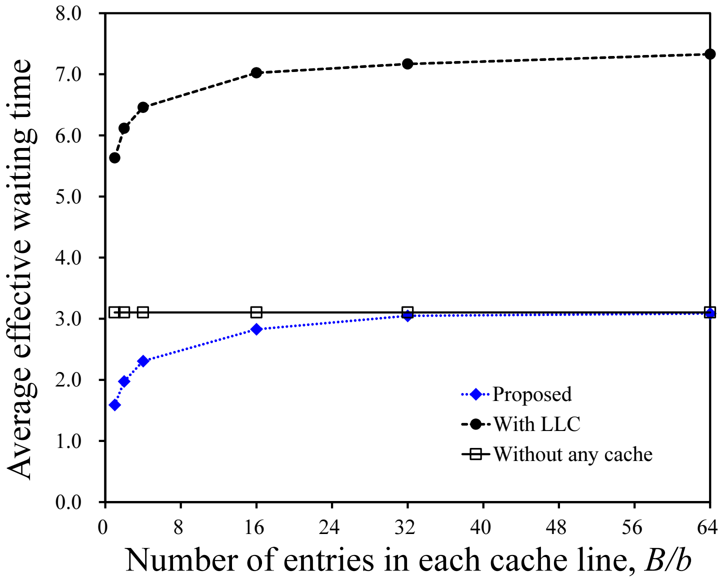


Figure 4.8: Average effective waiting time depending on number of entries in each cache line for different architectures.

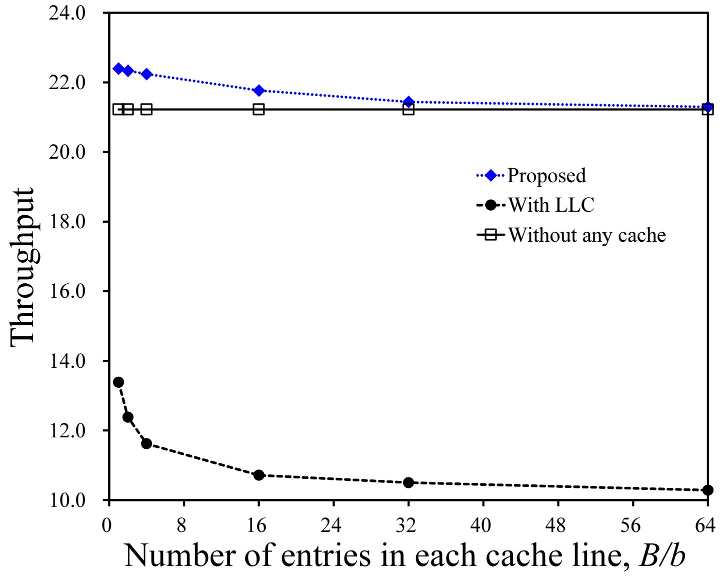


Figure 4.9: Throughput depending on number of entries in each cache line for different architectures.

with LLC, and the architecture without any cache, respectively; the set of numbers of entries in each cache line is considered as $\{1, 2, 4, 16, 32, 64\}$. This work observes that the proposed architecture reduces memory access latency by 58 % and 1.8 % and increases throughput by 104 % and 1 % with reducing the blocking probability 91 % and 18 %, compared to the architecture with LLC and the architecture without any cache, respectively, when $B/b = 32$, which is a typical value of the general-purpose processors. In addition, if the proposed architecture can use smaller cache lines with smaller B/b, the performance of proposed architecture can be improved.

4.4 Chapter summary

This chapter proposed a parallel memory system architecture that uses a 3D-stacked memory and on-chip private cache memories to reduce memory access latency in the existence of main memory parallelism for packet processing in network virtualization. The proposed architecture integrates the on-chip private cache memories of each CPU core, that is the L1 and L2 cache memories, with the off-chip 3D-stacked DRAM. Frequently accessed database entries in the 3D-stacked DRAM are copied and stored in the on-chip cache memories. This work explored the memory system architecture in terms of the integration of the on-chip cache memories with the off-chip 3D-stacked DRAM, including the architecture with LLC and the architecture without any cache. The queueing model-based simulation results observed that the proposed architecture reduced memory access latency by 58 % and 1.8 % and increased throughput by 104 % and 1 % with reducing the blocking probability 91 % and 18 %, compared to the architecture with LLC and the architecture without any cache, respectively, when $B/b = 32$, which is a typical value of the general-purpose processors. The memory performance would be improved by using the smaller cache lines with smaller B/b. The simulation results also observed that the on-chip shared LLC becomes a bottleneck due to the concentration of memory requests from multiple CPU cores as the traffic load increases, which implies that memory parallelism in on-chip cache memories is inevitable for packet processing in network virtualization when the main memory parallelism coexists.

Chapter 5

Parallel memory system architecture using interleaved 3D-stacked memory, private cache memory, and LLC slices

This chapter proposes a parallel memory system architecture that uses a 3D-stacked memory, on-chip private cache memories, and on-chip LLC slices to increase capacity and parallelism of the on-chip cache memories for packet processing in network virtualization. A part of the work in this chapter was presented in [168]. In modern multi-core CPUs, the on-chip LLC comprises the multiple LLC slices, each of which belongs to a CPU core and connected via a mesh or ring interconnection. A user or an operator of the general-purpose computers with such multi-core CPUs can assign some of the LLC slices to a specific VNF. In the proposed architecture, the LLC slices are used as the next level cache memory of the L2 cache memory, where a cache line evicted from the L2 cache memory is distributed one of the assigned LLC slices according to a memory address-based hash function so that CPU cores can access the LLC slices in parallel.

This work presents performance evaluation of the proposed architecture using the queueing-model simulations, in which the proposed architecture is compared with two reference architectures: one with the on-chip private cache memories, the on-chip shared LLC, and the off-chip 3D-stacked DRAM and one with the on-chip private cache memories and the off-chip 3D-stacked DRAM. In this chapter, these reference architectures are hereinafter referred to as the architecture with LLC and the architecture without LLC, respectively. The evaluation results observe that the proposed architecture reduces memory access latency by 62 % and 12 % and increases throughput by 108 % and 2 % with reducing the blocking probabilities by 96 % and 50 %, compared to the reference architectures with LLC and the architecture without LLC, respectively.

The rest of this chapter is organized as follows. Section 5.1 presents the proposed architecture. Section 5.2 describes the system modeling the proposed architecture and two reference architectures. Section 5.3 presents performance evaluation. Section 5.4 summarizes this chapter.

5.1 Proposed architecture

Figure 5.1 shows the proposed memory system architecture. It consists of a multi-core CPU, an FPGA, a 3D-stacked DRAM, a DRAM, and network interfaces. Each CPU core has its dedicated L1 and L2 cache memories. The on-chip LLC of the multi-core CPU comprises multiple LLC slices, each of which belongs to a CPU core and connected via an interconnection. The performance counter counts the statistics related to the CPU performance, such as the number of LLC misses and the number of elapsed core clock ticks. An FPGA connects a CPU and an off-chip 3D-stacked DRAM. An external DRAM is used as a packet buffer.

While most of the packet processing flow in the proposed architecture follows that in Sections 3.1 and 4.1, the proposed architecture has the on-chip LLC slices, into which a cache line evicted from the L2 cache memory is distributed. In the proposed architecture, each incoming packet is processed as follows.

(Step 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. The assigned CPU core reads the header information of the packet from the packet buffer

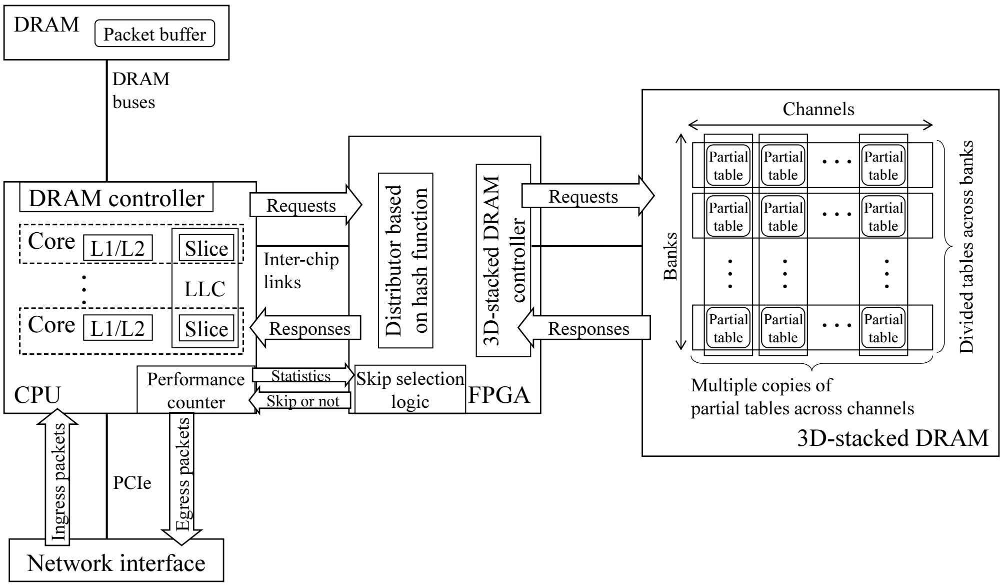


Figure 5.1: Overview of proposed architecture.

in the DRAM.

(Step 2) The CPU core issues memory requests to read table entries stored in the on-chip cache memories and the 3D-stacked DRAM in order to decide the next action for the packet.

(Step 3) The CPU core sends the packet including its payload from the packet buffer in the DRAM through the network interface.

This work mainly explores the integration of LLC slices with the on-chip private cache memories and off-chip 3D-stacked DRAM-based parallel main memory. Thus the packet processing step 2 in the above description is focused. In addition, performance of the packet processing steps 1 and 3 can be improved using the packet I/O acceleration methods such as DPDK, as described in Section 2.1. This work considers packet classification and database searching for examples of table lookup tasks. As with the explanations in Sections 3.1 and 4.1, the database stored in the 3D-stacked DRAM is split into several partial ones. Each partial database is located in the bank of a memory channel so that the original database is comprised of all the partial databases in a memory channel. The database, which is split into partial ones in a memory channel, is copied along with the memory channels so that every memory channel has the same database. This database placement allows the 3D-stacked DRAM to make the most of the memory-level parallelism and bank interleaving to increase memory performance for packet processing in network virtualization.

The FPGA accommodates the logic circuits for 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 hash-function-based distributor distributes issued memory requests into an appropriate memory channel-bank sets in the 3D-stacked DRAM so that the number of interleaved banks in each channel is minimum according to the bank selection rule presented in Section 3.2.

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 the off-chip 3D-stacked DRAM when there is a miss in the L2 cache memory. 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 by comparing the statistics of the LLC miss rate measured in the performance counter of the CPU with the threshold. This work explains this feature in Section 5.3.4.

5.2 System model

5.2.1 System model of proposed architecture

Figures 5.2 and 5.3 show the system models of the proposed architecture for two example cases. Figure 5.2 shows the case, in which all the CPU cores and LLC slices are assigned. Figure 5.3 shows the case, in which the numbers of assigned CPU cores and LLC slices are less than the numbers of CPU cores and LLC slices that the system has. 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 L1 and L2 cache memories. The on-chip shared LLC comprises multiple LLC slices, each of which has a queue. The queues in front of the 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. This work defines the number of assigned CPU cores and LLC slices to the application as C and L, where $0 \leq C \leq C_{\text{sys}}$ and $0 \leq L \leq L_{\text{sys}}$, respectively. Figure 5.2 shows the system model of proposed architecture where $C = C_{\text{sys}}$ CPU cores and $L = L_{\text{sys}}$ LLC slices in the system are assigned. Figure 5.3 shows system model of proposed architecture where $C < C_{\text{sys}}$ CPU cores and $L < L_{\text{sys}}$ LLC slices are assigned.

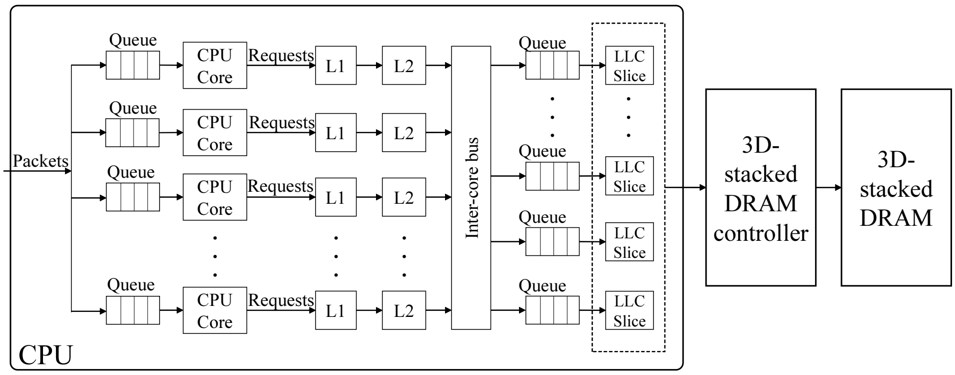


Figure 5.2: System model of proposed architecture, in which all queues, CPU cores, and LLC slices are assigned.

The on-chip L1 cache memory, L2 cache memory, and LLC slices 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 main memory system for database searching, such as for table lookup, is explained in Chapter 3. 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

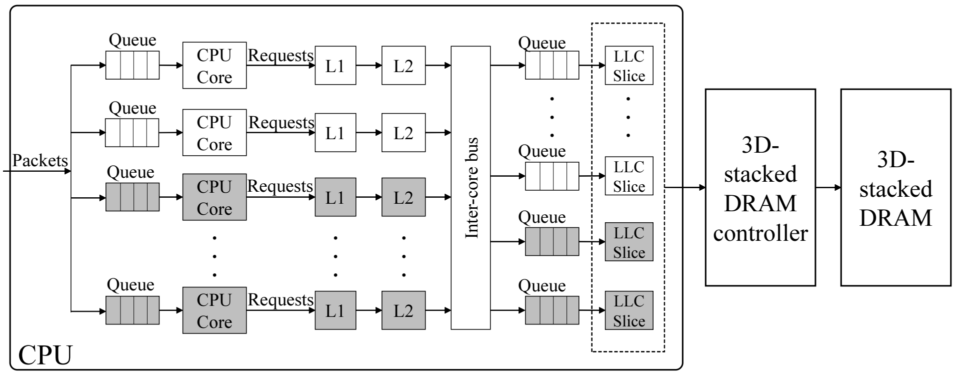


Figure 5.3: System model of proposed architecture, in which gray queues, CPU cores, and LLC slices are not assigned.

of 3D-stacked DRAM. The banks in the same memory channel can be interleaved, which enables the multiple partial tables in the same memory channel to be accessed efficiently. A request entering the queue is served at a memory channel-bank set in a FCFS policy. The memory channel-bank set to which a request transferred is selected so that the number of interleaved bank in each channel is minimum as described in Section 3.2.

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 memory of the CPU core at first. When there is a miss, the request accesses the L2 cache memory. The inter-core bus transfers a request that misses the L2 cache memory to the queue in front of an appropriate 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 slice is transferred to the 3D-stacked DRAM controller to wait for being distributed to one of memory channel-bank sets in the 3D-stacked DRAM.

Figure 5.4 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 5.3. 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, $\cdots$, 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, $\cdots$, R5 correspond to the first five incoming packets P1, P2, $\cdots$, 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 memory.

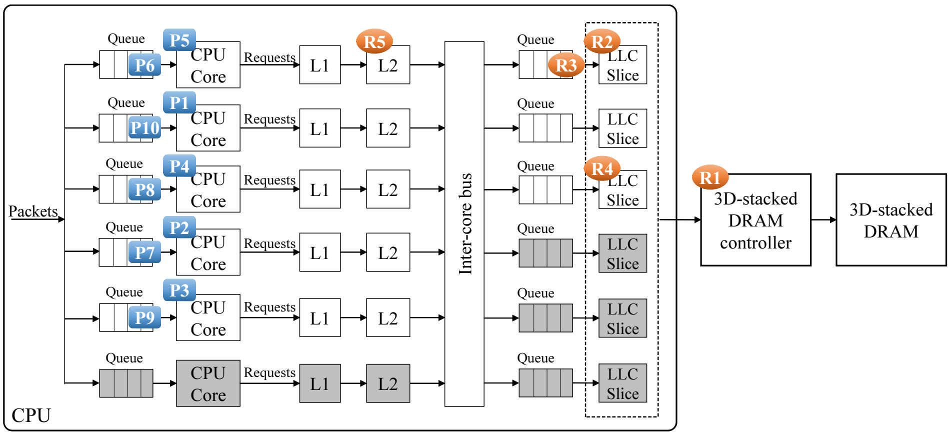


Figure 5.4: Example of system model of proposed architecture where $C_{\text{sys}} = 6$, $L_{\text{sys}} = 6$, $C = 5$, $L = 3$. There are ten packets P1, P2, $\cdots$, P10.

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 capacities of L1 cache memory, L2 cache memory, LLC slice, and 3D-stacked DRAM, respectively, each of which is given in the system model. The 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 memory from off-chip 3D-stacked DRAM. Then, the LRU cache line in the L1 or L2 cache memory of the corresponding CPU core is evicted to the L2 cache memory 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 memory as the most recently used cache line. When there is a miss, the LRU cache line in the L1 or the L2 cache memory is evicted to the L2 cache memory or LLC.

5.2.2 System model of reference architectures

Figures 5.5 and 5.6 show the system models of two reference architectures, which are compared with the proposed architecture: the architecture with LLC and the architecture without LLC. The devices that comprise the reference architectures are the same as those in the proposed architecture.

Figure 5.5 shows the system model of a reference architecture that has the on-chip physically shared LLC. The LLC is shared among all the CPU cores in the CPU. Memory capacity of the 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 Figures 5.2 and 5.3, except for how a request that misses L2 cache memory moves before it hits or misses the on-chip shared LLC. This work assumes that the on-chip 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 memory waits for the access to the on-chip shared LLC in the shared queue in front of the on-chip shared LLC.

Figure 5.6 shows the system model of the reference architecture without LLC. All the behavior of this system model is the same as the other system models in this work except for the on-chip LLC. A request that misses L2 cache memory is transferred to a common queue and enters the 3D-stacked DRAM controller as an FCFS policy.

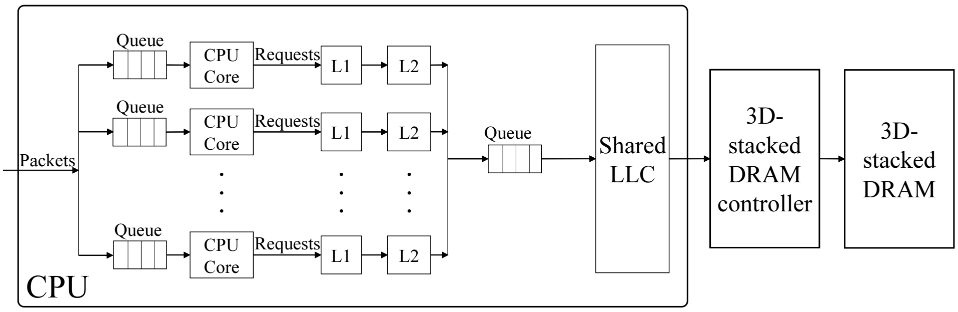


Figure 5.5: System model of reference architecture with LLC.

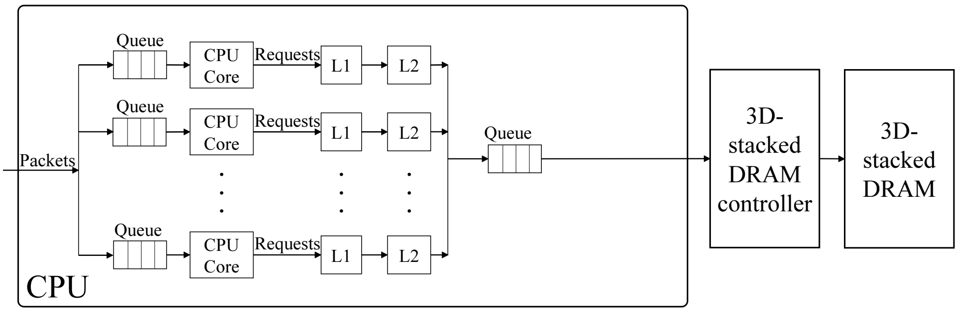


Figure 5.6: System model of reference architecture without LLC.

Figure 5.7 describes an example state of the system model of the reference architecture with LLC shown in Figure 5.5. 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 memories, an on-chip 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 5.4, there are ten incoming packets in the order of P1, P2, $\cdots$, 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 correspond to the packet. In other words, the five memory requests R1, R2, $\cdots$, R5 that correspond to the first five incoming packets P1, P2, $\cdots$, 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 memory 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 memory, and R5 is accessing the L1 cache memory. If R4 and R5 miss both the L1 and L2 cache memories, they will be transferred to shared LLC.

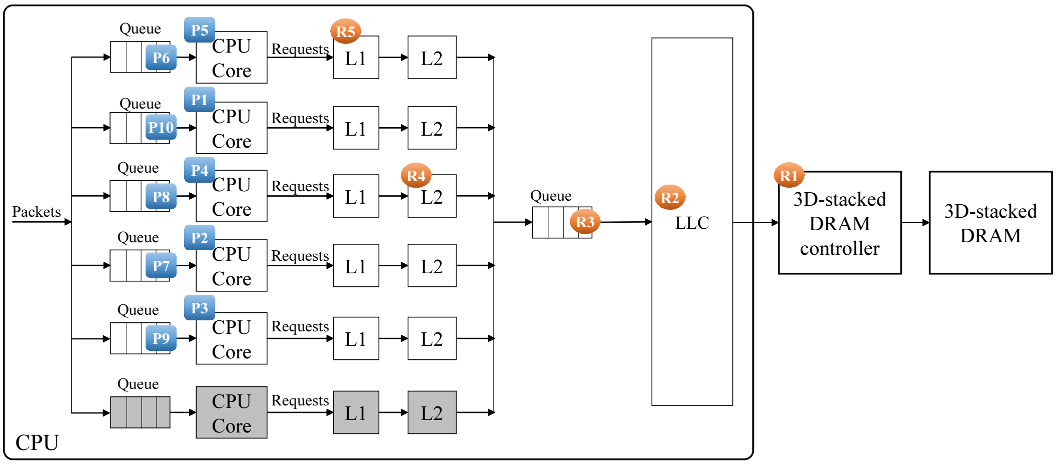


Figure 5.7: Example state of reference architecture with LLC, where $C_{\text{sys}} = 6$, $C = 5$. There are ten packets P1, P2, $\cdots$, P10.

For both system models that correspond to Figures 5.5 and 5.6, the system parameters C_{sys}, C, K, M_{L1}, M_{L2}, M_{3D}, B, and b are common with the description in Section 5.2.1. When the LRU cache line in the L2 cache memory is evicted, the cache line is transferred to the on-chip shared LLC in the system model that corresponds to Figure 5.5. For the system model that corresponds to Figure 5.6, the LRU cache line in the L2 cache memory is dropped when the LRU cache line in the L1 cache memory is evicted to the L2 cache memory.

5.3 Evaluation

5.3.1 Traffic model

In this work, the traffic model for the memory requests follows that presented in Chapter 4. In other words, a memory request arrives at the memory system based on a Poisson arrival process whose average rate is λ. The total number of table entries stored in the system is denoted as N_{ent}. This work assumes that the requested content follows a Zipf-like distribution, in which the normalized probability of requesting the ith most popular table entry is expressed as $\frac{\frac{1}{i^{\alpha}}}{\sum_{i=1}^{N_{\mathrm{ent}}} \frac{1}{i^{\alpha}}}$. The typical value of α is in the range of $0 < \alpha < 1$. As α increases, more requests are concentrated on a few most popular contents. This work sets the value of α as 0.83 [165] unless otherwise stated.

5.3.2 System assumption

This subsection introduces the assumption of the system model for the simplicity of the numerical simulations.

This work assumes 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 this work assumes 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}.

This work considers 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 [166] 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, this work assumes that the processing times in L1/L2 caches follow exponential distributions. According to [74], the access time to LLC slice varies depending on the distance between a CPU core and an LLC slice. In the simulations, this work assumes 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.

This work 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.

This work assumes that, the database that associate with multiple packet processing tasks in VNFs may not be accommodated in the on-chip caches due to a large number of subscribers or complex rules for searching and classification process. While this work takes the table lookup as an example of packet processing, the type of database in such VNFs are not limited to the lookup tables, and there can be other types of database 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 general-purpose computers in the evaluation. Without this assumption on the memory capacity of each cache and the number of table entries, the numerical simulations cannot finish within a practical time. For instance, if simulation parameters are set as $M_{\mathrm{L1}} = 64$ [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.

5.3.3 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 $|R_{\mathrm{b}}|$ 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 $P_{\mathrm{b}} = \frac{|R_{\mathrm{b}}|}{|R|}$. Throughput, λ_{e}, is defined by $\lambda_{\mathrm{e}} = \lambda(1 - P_{\mathrm{b}})$. For accepted request $r \in R \backslash R_{\mathrm{b}}$, this work defines t_r as the waiting time until it is processed by the CPU core. Thereby, this work defines W_{e} as the average effective waiting time by $W_{\mathrm{e}} = \frac{\sum_{r \in R \backslash R_{\mathrm{b}}} t_r}{|R| - |R_{\mathrm{b}}|}$. The sets of requests that hit the L1 cache memory, the L2 cache memory, and the LLC are represented by R_{L1},

R_{L2}, and R_{LLC}, respectively. Hit probabilities in the L1 cache memory, the L2 cache memory, and the LLC are defined by $P_{\mathrm{L1}} = \frac{|R_{\mathrm{L1}}|}{|R|}$, $P_{\mathrm{L2}} = \frac{R_{\mathrm{L2}}}{|R \backslash R_{\mathrm{L1}}|}$, and $P_{\mathrm{LLC}} = \frac{|R_{\mathrm{LLC}}|}{|R \backslash R_{\mathrm{L1}} \backslash R_{\mathrm{L2}}|}$, respectively.

This work considers 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 bank interleaving are $\mu = \mu_2$ and μ_1, 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. This work considers the IP addresses lookup based on DIR-24-8-BASIC [13] algorithm for the benchmark of the packet processing, where the size of each entry is 2 byte. Thereby, this work sets $b = 2$ [B]. This work also sets 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], $C_{\mathrm{sys}} = 28$, $L_{\mathrm{sys}} = 28$, $\rho = 0.7$, $N_{\mathrm{ent}} = 2 \times 10^4$, $\mu_{\mathrm{L1}} = 100$, $\mu_{\mathrm{L2}} = 50$, $\mu_{\mathrm{LLC}} = 10$, $\mu = \mu_1 = 1$, $\mu_2 = 0.7$, $S = 32$. 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.

5.3.4 Numerical results

This work presents the numerical simulation results of three system models, each of which corresponds to the following architecture: (a) proposed architecture, (b) reference architecture with LLC, and (c) reference architecture without LLC. In this subsection, this work labels these three as *Proposed*, *With LLC*, and *Without LLC*.

Figures 5.8, 5.9, and 5.10 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 $N_{\mathrm{ent}} = 2\times10^4$; the set of numbers of entries in each cache line is considered as $\{1, 2, 4, 16, 32, 64\}$. This work observes 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 memory 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 memory and degrades the system performances in terms of the blocking probability, the average effective waiting time, and the throughput. This work observes that the proposed architecture outperforms the other two architectures in terms of each considered aspect. Today's typical general-purpose computer systems have 64-byte cache lines, which corresponds to $B/b = 32$. When $B/b = 32$, the proposed architecture reduces memory access latency by 62 % and 12 % and increases throughput by 108 % and 2 % with reducing the blocking probability by 96 % and 50 %, compared to the architecture with LLC and the architecture without LLC, respectively.

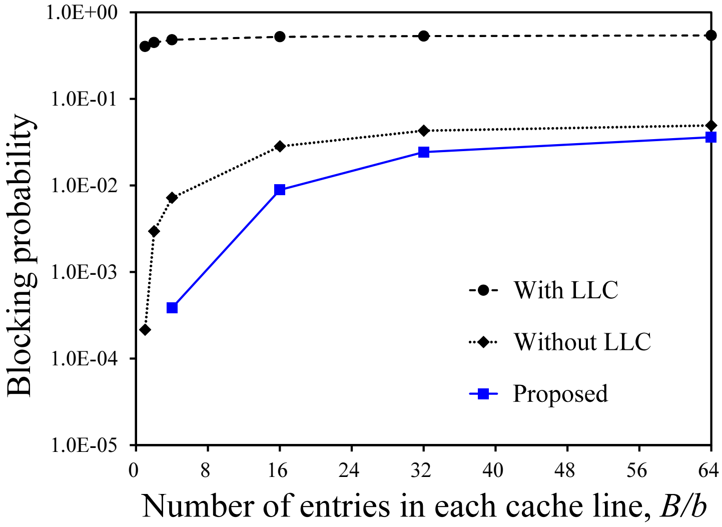


Figure 5.8: Blocking probability depending on number of entries in each cache line.

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 in the similar method presented in Section 3.5.3. For instance, let the system have a lookup table for IP addresses lookup, where table entry of each IP address is allocated in a flat manner so that every lookup for a packet finishes with one memory access. Typical latency of single memory access to

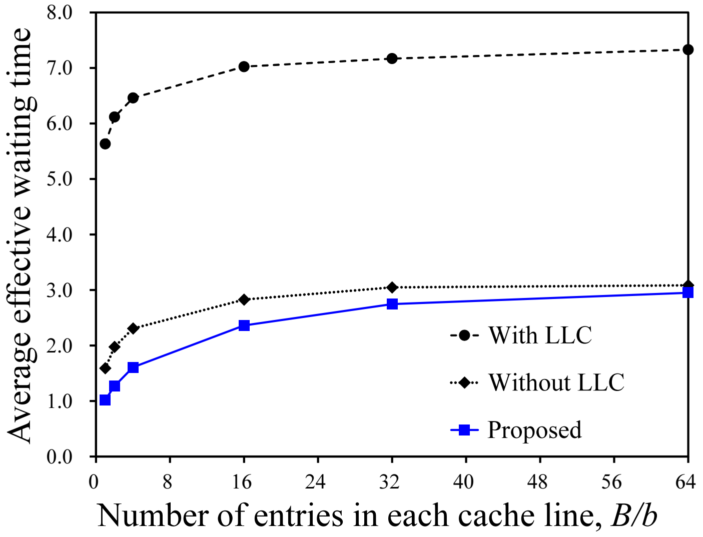


Figure 5.9: Average effective waiting time depending on number of entries in each cache line.

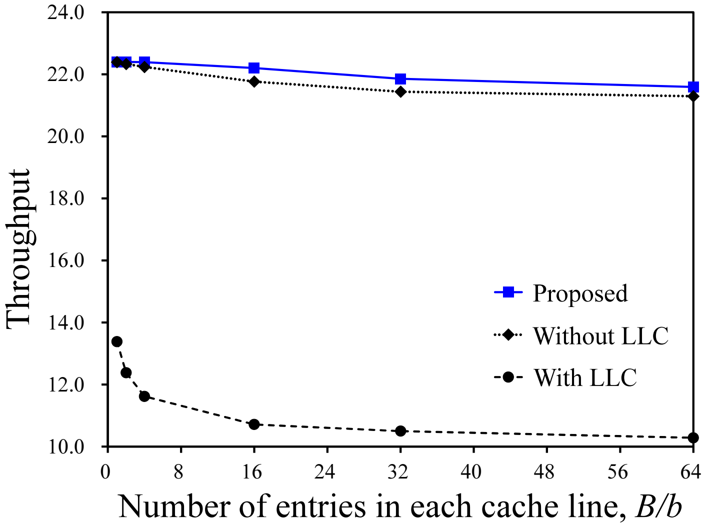


Figure 5.10: Throughput depending on number of entries in each cache line.

HMC is usually between 100-180 [ns] with an average of 125 [ns], according to [156, 157]. Thus this work assumes that the service rate of the 3D-stacked DRAM, μ, is 8 M services per second. As well as the same configuration for Figures 5.8, 5.9, 5.10, this work sets $C = 28$, $L = 28$, $S = 32$, $K = 100$, $\rho = 0.7$, $\alpha = 0.83$, $B/b = 32$, and $N_{\text{ent}} = 2 \times 10^4$. Based on these configurations, this work calculates the throughput of processing 64 [B] packets and latency of single memory access. The calculated throughput and latency of the proposed 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 5.11, 5.12, and 5.13 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 $N_{\text{ent}} = 2 \times 10^4$; the set of numbers of assigned LLC slices, L, is considered as $\{1, 4, 8, 16, 28\}$. This work observes that, as the number of assigned LLC slices increases, since the 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 5.14, 5.15, and 5.16 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, respectively, where $B/b = 32$, $\rho = 0.7$, and $N_{\text{ent}} = 2 \times 10^4$; 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 16 and 28, the proposed architecture outperforms the architecture without LLC.

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

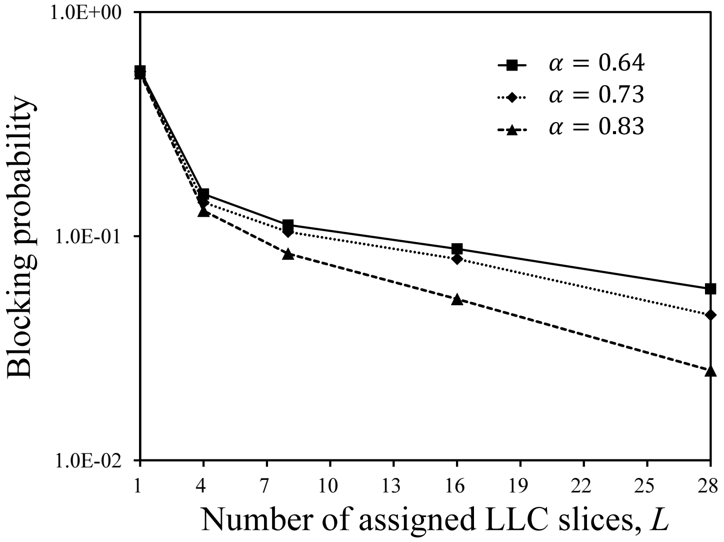


Figure 5.11: Blocking probability depending on number of assigned LLC slices.

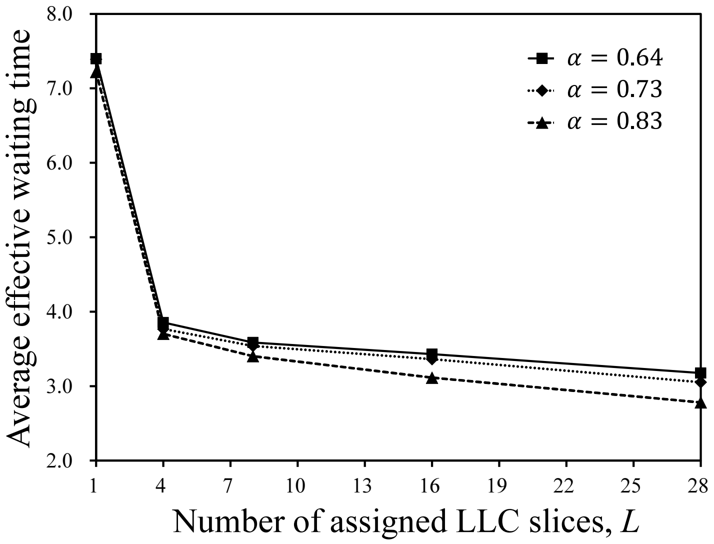


Figure 5.12: Average effective waiting time depending on number of assigned LLC slices.

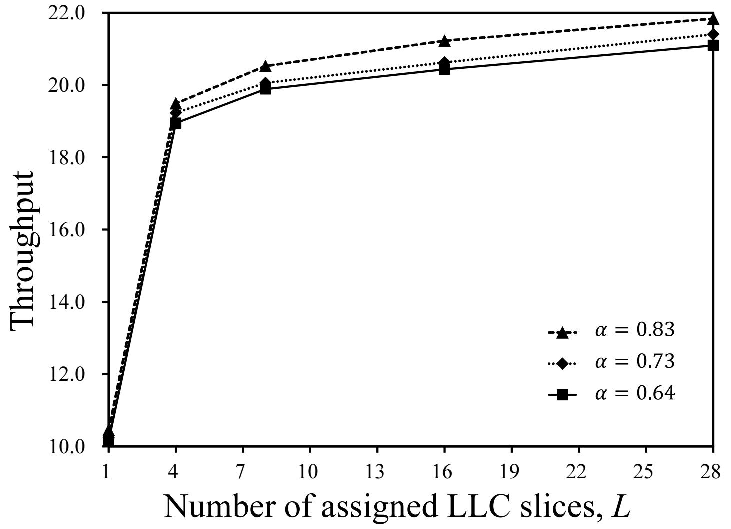


Figure 5.13: Throughput depending on number of assigned LLC slices.

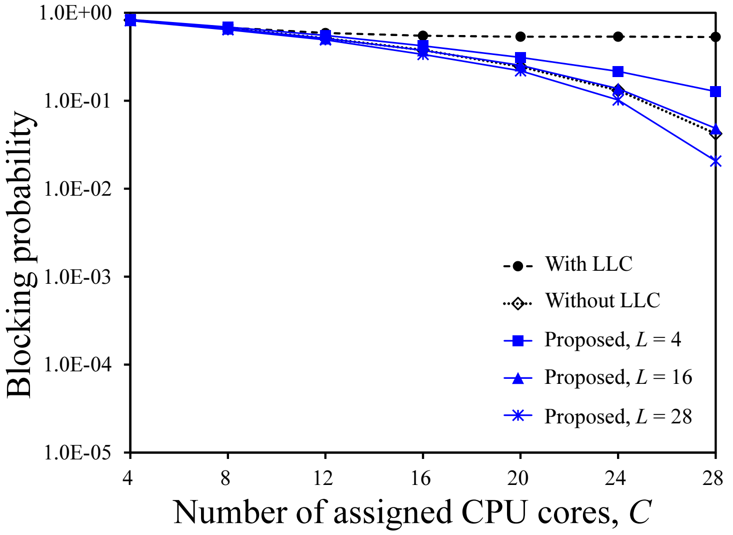


Figure 5.14: Blocking probability depending on number of assigned CPU cores $B/b = 32$.

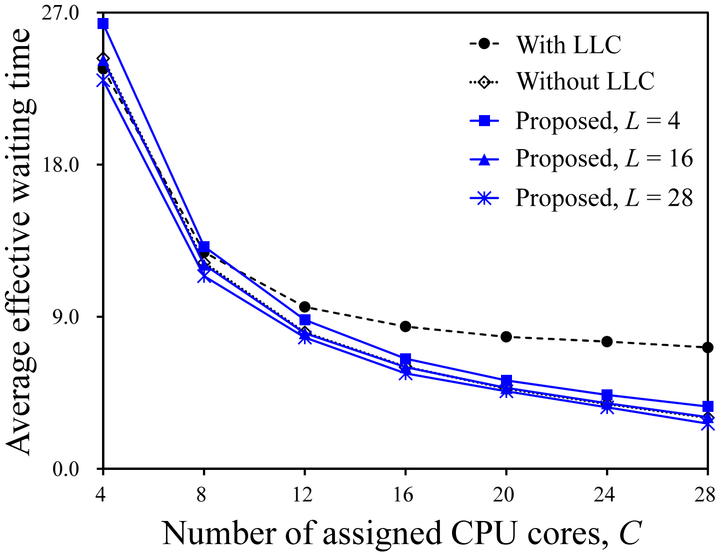


Figure 5.15: Average effective waiting time depending on number of assigned CPU cores $B/b = 32$.

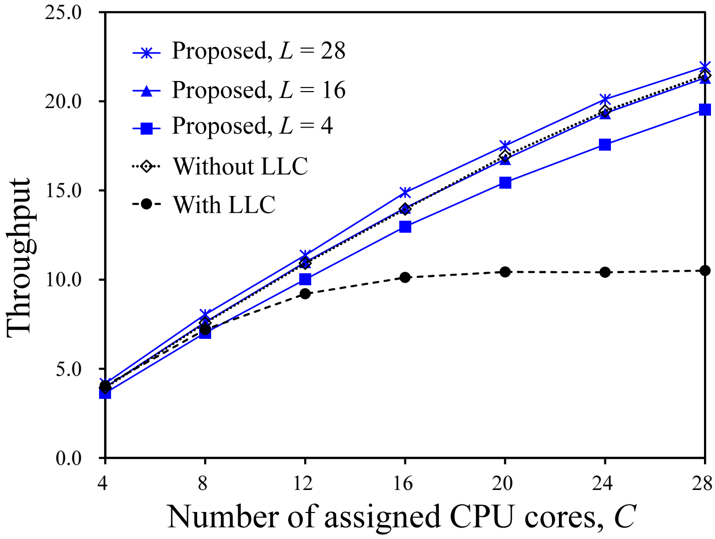


Figure 5.16: Throughput depending on number of assigned CPU cores $B/b = 32$.

Figures 5.17, 5.18, 5.19, and 5.20 show the blocking probabilities, average effective waiting times, throughputs, and hit rates of each level of cache memory 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 $\{10^3, 10^4, 2 \times 10^4, 3 \times 10^4, 4 \times 10^4, 5 \times 10^4, 6 \times 10^4\}$. This work observes 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 cache memories decreases, which degrades the efficiency of the cache. As a result, the hit rate in each level of cache memory decreases, and the system perform worse in terms of each considered aspect.

Figures 5.17, 5.18, and 5.19 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_{ent} is small, or $N_{\text{ent}} = 10^4$, the proposed architecture significantly outperforms the architecture without LLC. The benefit of proposed architecture decreases as the value of N_{ent} increases; when the value of N_{ent} is greater than one point, the architecture without LLC slightly outperforms 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 [169, 170], which allows the requests that miss the L2 cache memory 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 [171–176].

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.

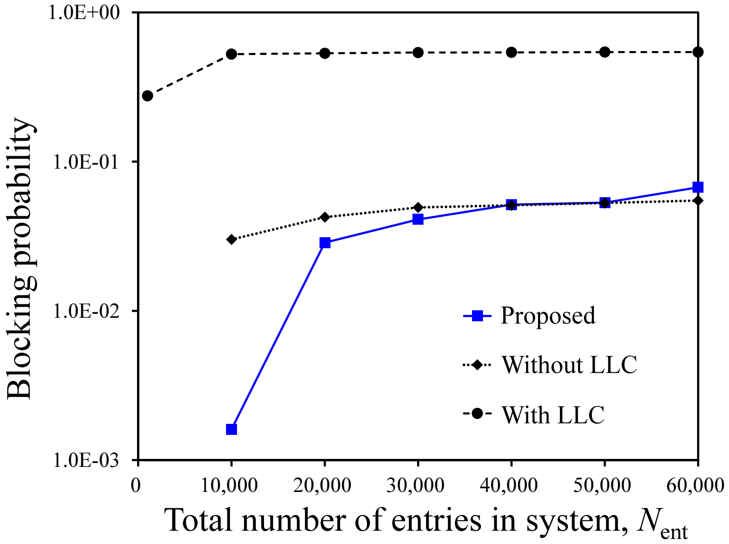


Figure 5.17: Blocking probability depending on total number of entries.

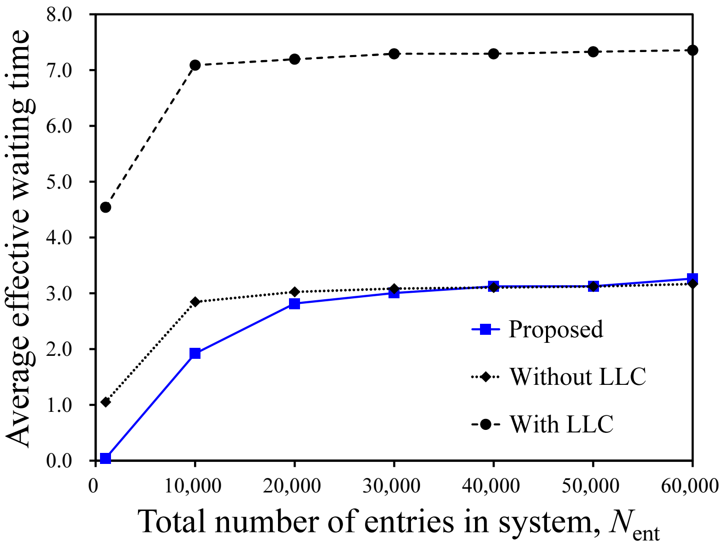


Figure 5.18: Average effective waiting time depending on total number of entries.

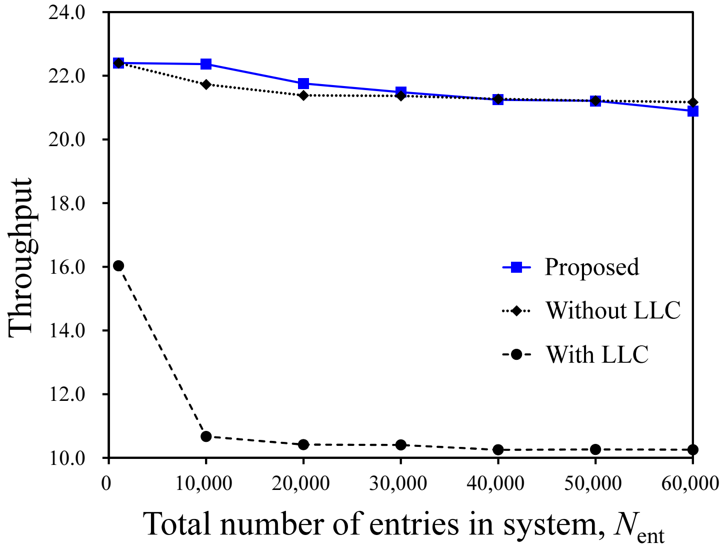


Figure 5.19: Throughput depending on total number of entries.

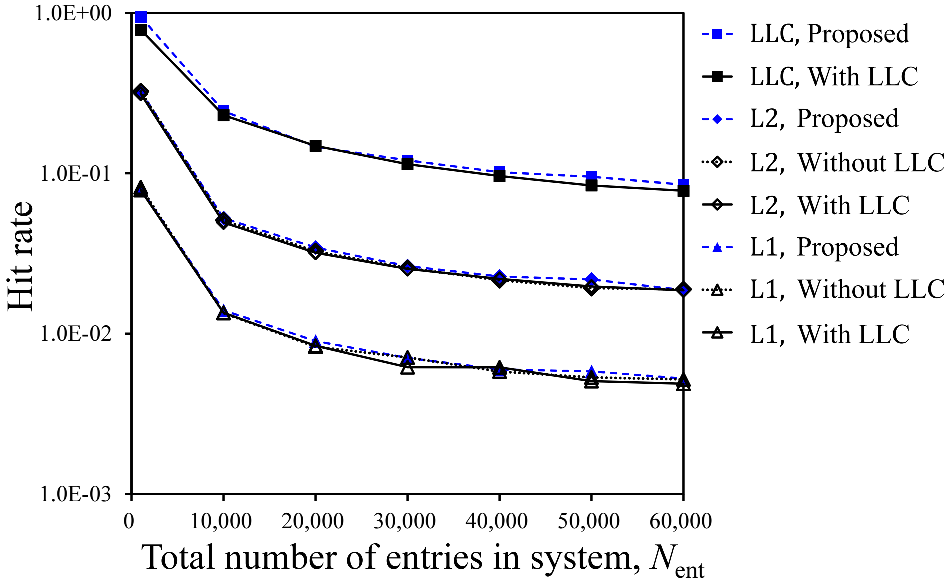


Figure 5.20: Hit rate of each cache level depending on total number of entries.

5.4 Chapter summary

This chapter proposed a parallel memory system architecture that uses a 3D-stacked memory, on-chip private cache memories, and on-chip LLC slices to increase capacity and parallelism of the on-chip cache memories for packet processing in network virtualization. The on-chip shared LLC comprises multiple LLC slices, each of which belongs to a CPU core and is connected to the other CPU cores via a ring or mesh interconnection. An operator of a general-purpose computer system that has the CPU with LLC slices can assign some of the LLC slices to a specific application. The proposed architecture integrates the LLC slices with the on-chip private cache memories and the off-chip 3D-stacked DRAM-based parallel main memory. The proposed architecture also contains the skip selection logic that determines if a memory request that misses the L2 cache memory skips the LLC based on the statistics of performance counters. The queueing model-based simulation results observed that the proposed architecture reduces memory access latency by 62 % and 12 % and increases throughput by 108 % and 2 % with reducing the blocking probability by 96 % and 50 %, compared to the architecture with LLC and the architecture without LLC, respectively. When more CPU cores and LLC slices are assigned, the performance of proposed architecture increases. The evaluation results also implied that the benefit of LLC slices decreases as the database size in the system is larger than one point; the architecture without LLC slightly outperforms the proposed architecture. For such a case, the skip selection logic in the proposed architecture can select skipping the LLC to avoid performance degradation.

Chapter 6

Parallel memory system architecture using network of interleaved 3D-stacked memories

This chapter proposes a parallel memory system architecture that uses a network of 3D-stacked DRAMs to increase throughput and reduce accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses [177]. Packets that enter the memory network receive packet processing at each 3D-stacked DRAM without going back and forth between the memories and the processors. Each 3D-stacked DRAM has several DRAM layers on top of a logic die. The logic die consists of a logic for memory accesses and device interfaces and a user-defined programmable logic, in which each packet processing task is placed. If a packet processing task needs to search a database, the database is stored in the same 3D-stacked DRAM as the task is allocated. The database is replicated and allocated in every memory channel to leverage memory parallelism of 3D-stacked DRAMs. For a packet processing task that requires more memory accesses than the other tasks, additional 3D-stacked DRAMs in the memory network are assigned for the task.

The evaluation results observe that the proposed architecture reduces the blocking probability by issuing the next memory request inside the 3D-stacked DRAM just after the previous memory access, instead of allowing the arrivals of incoming packets during the data transfers between the memory and processor. Consequently, the proposed architecture increases the throughput and reduces the accumulated latency when there are multiple packet processing tasks, compared to the conventional architecture with 3D-stacked DRAM-based parallel memory, where every memory access requires data transfers between the processors and memories. The proposed architecture also reduces the blocking probability and latency by assigning more 3D-stacked DRAMs for a packet processing task that requires more memory accesses than the other tasks.

The rest of this chapter is organized as follows. Sections 6.1 presents the proposed architecture. Section 6.2 describes the system modeling. Section 6.3 presents numerical results. Section 6.4 summarizes this chapter.

6.1 Proposed architecture

Figure 6.1 shows the proposed memory system architecture for packet processing in memory network. The proposed architecture comprises network interfaces, an FPGA, a CPU, DRAMs, and a memory network that consists of 3D-stacked DRAMs. The FPGA interfaces the network interfaces and handles the ingress packets and egress packets. The FPGA also parses the ingress packets and classifies them to distribute the packets to the CPU or the memory network according to the packet content. The CPU and the DRAM execute the control plane processing. The FPGA and the memory network execute the data plane processing. Each 3D-stacked DRAM has an interface logic, a programmable custom logic, and multiple memory channels, each of which has several banks. In the 3D-stacked DRAM, the custom logic accesses memory channels. The interface logic of a 3D-stacked DRAM has up to four connections, and an additional 3D-stacked DRAMs can be chained to the 3D-stacked DRAM. Each packet processing task of a VNF is programmed to the custom logic of each 3D-stacked DRAM in advance. If a packet processing task needs to search a database, the database is placed in every memory channel of the same 3D-stacked DRAM where the packet processing tasks are allocated. The packet processing tasks of the VNF are allocated to the 3D-stacked DRAMs so that the original packet processing flow comprises the chain of the 3D-stacked DRAMs. The FPGA also has some packet processing tasks in the rest of the logic area when necessary.

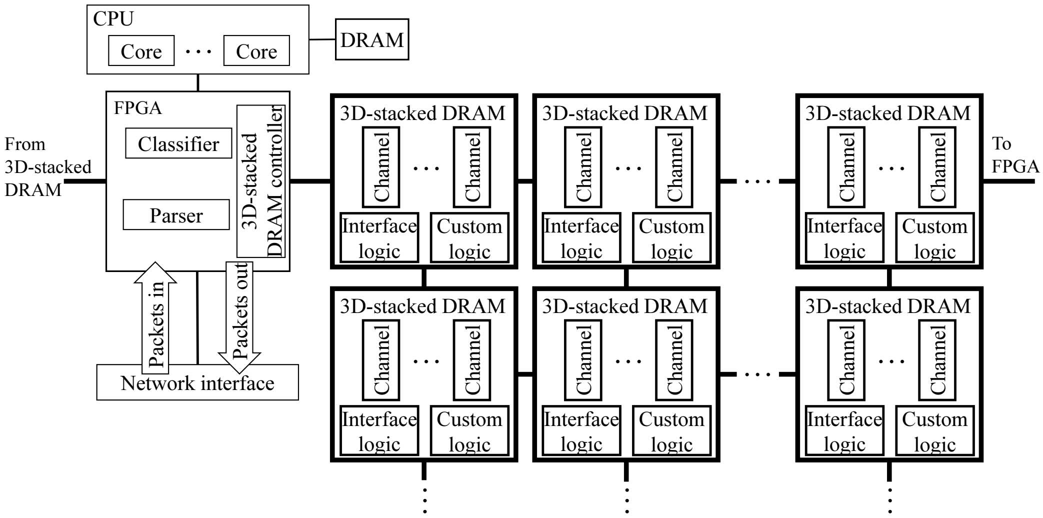


Figure 6.1: Proposed architecture.

When a packet arrives at the network interface, the packet is processed as follows.

(Step 1) The packet received at the network interface is transferred to the FPGA and is parsed by the packet parser function in the FPGA. Then the classifier logic determines whether the packet should be destined to the memory network or the CPU, according to the packet content. When the packet is for the control plane processing for such as a routing protocol, a monitoring protocol, and a management protocol, the packet is destined to the CPU, and the CPU executes application software for the control plane processing. When the packet is for the data plane processing, the packet is destined to the first 3D-stacked DRAM of the memory network.

(Step 2) The packet that enters the first 3D-stacked DRAM receives the first packet processing task. Each 3D-stacked DRAM corresponds to a single packet processing task, which is executed by the custom logic of the 3D-stacked DRAM. The custom logic issues the required number of memory requests to access the memory channels of the 3D-stacked DRAM to complete the programmed packet processing task for the packet. When the packet processing task for the packet is completed in the 3D-stacked DRAM, the next action for the packet is determined by the custom logic. If the packet needs to receive the next packet processing task, the packet is transferred to one of the adjacent 3D-stacked DRAMs in which the next packet processing task is programmed as described in step 3. If the packet does not have any further packet processing task, the packet is transferred to the FPGA.

(Step 3) The packet from the previous 3D-stacked DRAM enters the second and following 3D-stacked DRAMs and receives the corresponding packet processing task that each 3D-stacked DRAM has. As with step 2, the custom logic of each 3D-stacked DRAM issues memory requests and determines the next action of the packet. If there is no more packet processing task for a packet, the packet is transferred to the FPGA. If there is further packet processing task for a packet, the packet is transferred to one of the adjacent 3D-stacked DRAMs that have the task.

(Step 4) When packet processing in the CPU or the network of 3D-stacked DRAMs is completed, the packet is returned to the FPGA from the CPU or the memory network. Then, the packet is transferred to the network interface and transmitted from the system. If there is any further packet processing task, the FPGA processes the packet before the packet is transferred to the network interface.

This work focuses on the data plane packet processing as it requires high memory performance. Thus, the description above mainly presents the processing flow of the data plane packet processing. Furthermore, this work mainly considers the packet processing at step 2 and 3, which directly relate to memory performance in terms of memory parallelism and memory access latency. Therefore, Section 6.2 models the behavior of the memory network of 3D-stacked DRAMs.

This work assumes that there are a sufficient number of 3D-stacked DRAMs in the architecture to allocate the packet processing tasks of VNFs. For example, if there are four packet processing tasks, four 3D-stacked DRAMs are horizontally cascaded and connected to the FPGA. Each of the allocated packet processing tasks can also be allocated to vertically cascaded 3D-stacked DRAMs in addition to the 3D-stacked DRAM to which the packet processing task is allocated. Figure 6.2 depicts an example situation, where four packet processing tasks and data are allocated to each 3D-stacked DRAM, and additional 3D-stacked DRAMs are allocated for packet processing tasks 2 and 3. The grey 3D-stacked DRAMs are not used, whereas their interface logic remains active so that packets can pass through them when necessary.

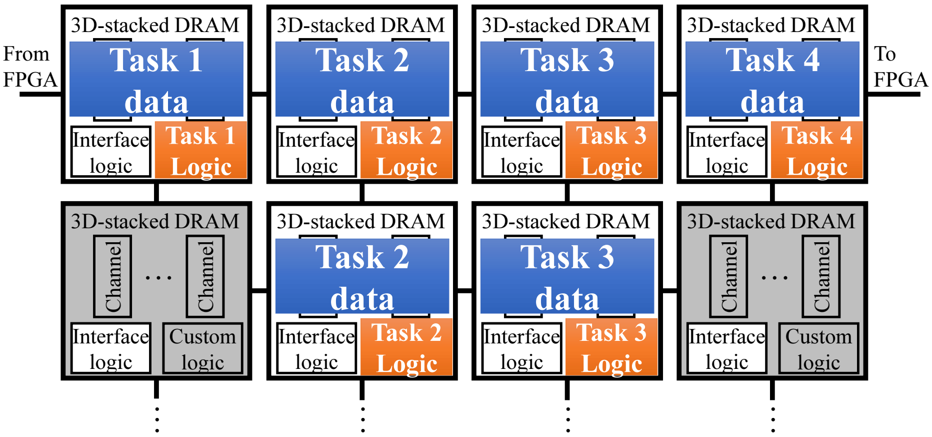


Figure 6.2: Example situation of proposed architecture.

6.2 System model

6.2.1 System model of memory network

This work considers the system model for the memory network of 3D-stacked DRAMs in the proposed architecture. The memory network consists of multiple 3D-stacked DRAMs, each of which is connected via horizontal links and vertical links. Figure 6.3 shows an overview of the system model for the memory network of 3D-stacked DRAMs in the proposed architecture, where there are LM 3D-stacked DRAMs in the memory network. For the following description, this work defines the four directions in the memory network as depicted in Figure 6.3; north, east, south, and west. The maximum number of 3D-stacked DRAMs in the network in an east-west direction and a north-south direction are represented by M and L, respectively. Each 3D-stacked DRAM in the memory network is denoted as 3D-stacked DRAM (i, j) for $i \in [1, L]$ and $j \in [1, M]$. Note that not all the LM 3D-stacked DRAMs are always active and have packet processing tasks; usually some of the 3D-stacked DRAMs are active and the others are vacant as depicted in Figure 6.2.

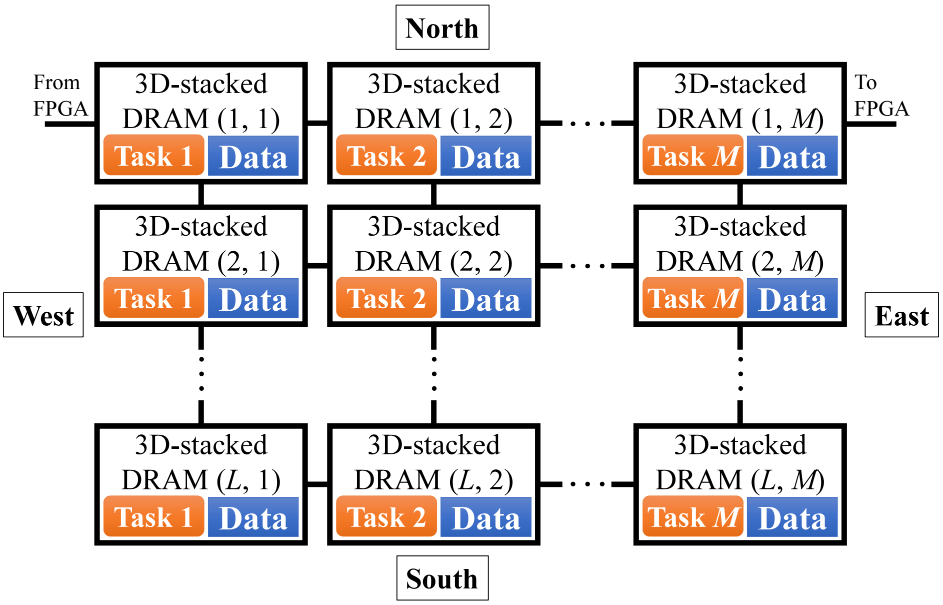


Figure 6.3: Overview of system model.

The packet processing tasks for a VNF are distributed and allocated to each 3D-stacked DRAM. Each 3D-stacked DRAM has a packet processing task, and its associated data is allocated in every memory channel in the 3D-stacked DRAM. A packet that enters the memory network goes through the memory network from west to east and receives the packet processing tasks one by one from each 3D-stacked DRAM. There may be a packet processing task that is allocated to multiple 3D-stacked DRAMs in a north-south direction to increase the memory parallelism, depending on the number of required memory accesses to process a packet.

This work assumes that each packet processing task and its associated data are firstly allocated to the northmost 3D-stacked DRAMs $(1, j)$, where $j \in [1, M]$. Additionally, this work assumes that, if the jth task and its associated data are to be allocated to additional 3D-stacked DRAMs, the 3D-stacked DRAM (i, j) is selected in an increasing order of i, where $i \in [2, L]$. This work defines the required number of memory accesses to process a packet for the jth packet processing task by ω_j, where $j \in [1, M]$. This work introduces the one-way packet transfer latency between the FPGA and the 3D-stacked DRAM, which is denoted as τ_{tr}.

6.2.2 System model of 3D-stacked DRAM

Figure 6.4 shows the system model of each 3D-stacked DRAM in the memory network. Each 3D-stacked DRAM has three input interfaces and their associated queues and three output interfaces and their associated queues. The three input interfaces correspond to the inputs from west, north, and south, and their associated queues are denoted as queue IW (input from west), queue IN (input from north), and queue IS (input from south), respectively. Also, the three output interfaces correspond to the outputs to east, south, and north, and their associated queues are denoted as queue OE (output to east), queue OS (output to south), and queue ON (output to north), respectively. Packets that enter these input and output queues are served in an FCFS policy.

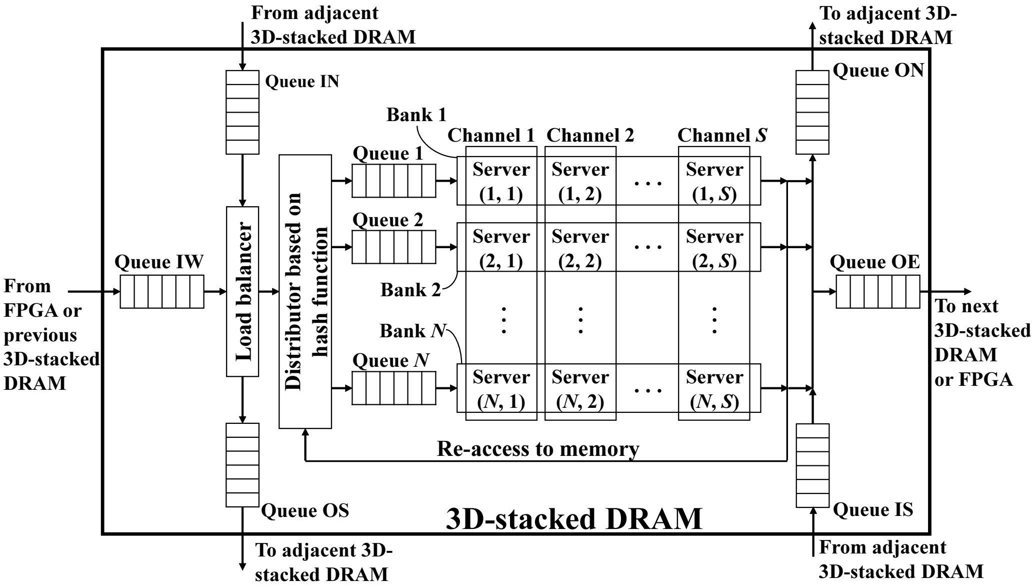


Figure 6.4: System model of 3D-stacked DRAM.

Each 3D-stacked DRAM has S memory channels, which can be accessed in parallel. Each memory channel has N banks, which can be accessed efficiently using bank interleaving technique. Therefore, each channel-bank set behaves as a server and is denoted as (n, s), where $n \in [1, N]$ and $s \in [1, S]$. A server that belongs to bank n has queue n, which is served according to the FCFS policy. If the allocated packet processing task has its associated data, the data is stored in the 3D-stacked DRAM so that incoming packets can access the data in parallel as presented in [133]. The data is equally split into N partial data and stored in a memory channel. Then the data in the memory channel is copied to the other memory channels of the same 3D-stacked DRAM. Consequently, each server has its partial data.

Packets may enter a 3D-stacked DRAM from the west or north. The west corresponds to the FPGA or the 3D-stacked DRAM that has the previous packet processing task, and the north corresponds to the vertically adjacent 3D-stacked DRAM in the upper direction that has the same packet processing task as the current 3D-stacked DRAM. The incoming packet first enters the load balancer, which determines whether the packet is processed in the current 3D-stacked DRAM or in the vertically adjacent 3D-stacked DRAM in the lower direction if any. If the packet is determined to be processed in the current 3D-stacked DRAM, the packet enters the distributor based on a hash function. If the packet is destined to the southbound, the packet enters the queue OS and waits for the transfer to the adjacent 3D-stacked DRAM. When the packet is transferred to and arrives at the adjacent 3D-stacked DRAM, the packet is enqueued in the queue IN of the adjacent 3D-stacked DRAM and processed in the same way as the current 3D-stacked DRAM.

The packet from the load balancer is processed by the packet processing task that is allocated in the custom logic. The logic issues the required number of memory requests for the packet processing task. The issued memory requests are processed by the memory subsystem that consists of a hash-function-based distributor, NS servers, and N queues, as presented in [133]. The memory request that enters the hash-function-based distributor is distributed to one of the queues associated with each bank of the memory channels in the 3D-stacked DRAM. The distribution to queue n is conducted according to the packet content so that the packet is destined to the appropriate servers that have corresponding partial data or memory address. The maximum number of memory requests that can be accommodated in the N queues and all the servers is K, where $K \geq NS$. A memory request is blocked if the number of memory requests in the memory subsystem exceeds K. A packet whose associated memory request is blocked is discarded. Memory resources for N queues are shared under the condition that the total number of accommodated memory requests in the memory subsystem does not exceed K, which means that, in the worst case, $K - S$ memory requests are waiting in one particular queue and S memory requests are served by the corresponding S servers. The sizes of queue IN, queue IW, queue IS, queue OS, queue OE, and queue ON are K.

The service rate of server (n, s) is the memory access rate of bank n at memory channel s. When more than one server at the same memory channel s is serving memory requests, these servers perform bank interleaving; otherwise there is no bank interleaving. When there are w banks that perform bank interleaving at the memory channel s, this work calls that the w-degree bank interleaving. Consequently, there is no bank interleaving when $w = 1$.

When a server finishes its service, the response to the memory request is returned to the custom logic. If the packet processing task is completed by receiving the response, the corresponding packet departs the memory subsystem and its next action is determined out of the three: returns to the FPGA, enters to queue OE, and enters to queue ON. If the packet has no more packet processing task in the memory network, the packet returns to the FPGA. If the packet needs to receive the next packet processing task, the packet is destined to queue OE or queue ON. When there is an adjacent 3D-stacked DRAM to the east, the packet is enqueued in queue OE and waits for the transfer to the east 3D-stacked DRAM. When there is no adjacent 3D-stacked DRAM to the east, the packet is destined to the queue ON to waits for the transfer to the north 3D-stacked DRAM. The next action of an incoming packets from queue IS is determined in the same way as the packet that departs the memory subsystem.

If the packet processing task allocated in the custom logic needs to access the memory again, the additional memory request is distributed to the appropriate queue by the hash-function-based distributor and waits for the next service. If the packet processing task is completed as a result of the memory access, the corresponding packet departs the memory subsystem, and its next action is determined as with the description above.

6.3 Numerical results

6.3.1 Performance metrics and evaluation settings

This work evaluates the blocking probability, throughput, and average effective waiting time as performance metrics of the proposed architecture. Let T represent a set of packets that enter the memory network during a certain period of time. The total number of packets in T is denoted by $|T|$. The number of discarded packets in the memory network during the period is represented by $|T_b|$, where T_b denotes the set of discarded packets. This work defines the blocking probability P_b as a probability that a packet is discarded in the memory network, which is represented by $P_b = \frac{|T_b|}{|T|}$. Consequently, throughput λ_e is defined by $\lambda_e = \lambda(1-P_b)$. For processed packet without being discarded in the memory network $t \in T \setminus T_b$, this work defines its waiting time to be returned to the FPGA by τ_t. This work introduces average effective waiting time W_e, which is defined by $W_e = \frac{\sum_{t \in T \setminus T_b} \tau_t}{|T| \setminus |T_b|}$. This work introduces a traffic load ρ, which is defined by $\rho = \frac{\lambda}{S\mu}$.

This work assumes that a packet departs the FPGA and arrives at the queue IW of the first 3D-stacked DRAM of the memory network following a Poisson arrival process with average rate of λ. Since the processing time of DRAM depends on the DRAM specific mechanism, such as precharge before loading another row, this work assumes that the service rate of server (n, s) follows an exponential distribution with average service rate of μ_w for w-degree bank interleaving, where $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_N$ with $w\mu_w \geq \mu_1$.

In the numerical simulations, this work considers that each 3D-stacked DRAM in the memory network has $S = 32$ memory channels with $N = 2$ banks. Thus, the service rates of each server in the 3D-stacked DRAM with and without bank interleaving are represented as μ_2 and μ_1, respectively. This work sets $\mu = \mu_1 = 1$ and $\mu_2 = 0.7\mu_1 = 0.7$. Consequently, the average time required for a single DRAM access without bank interleaving is considered as $\frac{1}{\mu} = 1$. This work also sets $K = 100$ as with the works in [133]. According to the works in [157, 178, 179], the average time required to access the DRAM inside an HMC is reported to be around 40 ns, and the typical time required for a single access to an HMC is reported to be between 100–180 ns with an average of 125 ns, when there is no load. Based on these observations, this work assumes that the required time for a round-trip packet transfer between the FPGA and the 3D-stacked DRAM is around 80 ns. Consequently, this work assumes that an one-way packet transfer latency between the FPGA and the 3D-stacked DRAM, τ_{tr}, is around 40 ns, which is approximately equal to the DRAM latency. This work sets $\tau_{\mathrm{tr}} = 1$.

6.3.2 Performance dependency on number of packet processing tasks and traffic load

This work considers a memory network with $M = \{1, 2, 4, 8\}$ and $L = 1$ as depicted in Figure 6.5. Each of the M 3D-stacked DRAMs has a packet processing task, whose required numbers of memory accesses are $\omega_i = 2$ for $i \in [1, M]$.

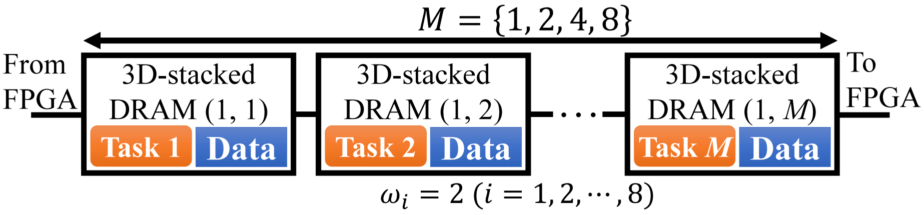


Figure 6.5: Memory network for evaluation of dependency on number of packet processing tasks and traffic load.

This work introduces a reference architecture that consists of an FPGA and $M = \{1, 2, 4, 8\}$ 3D-stacked DRAMs, as shown in Figure 6.6. The reference architecture is introduced to model a conventional memory system architecture, where the packet processing tasks are executed in the FPGA, and every memory access is accompanied by the data transfer between the FPGA and 3D-stacked DRAMs. The 3D-stacked DRAMs in the reference architecture has the same memory subsystem as presented in Figure 6.4, where there are a hash-function-based distributor, N queues, and NS servers. Each 3D-stacked DRAM corresponds to one of the M packet processing tasks, each of which requires two memory accesses, where $\omega_i = 2$, to complete the task.

Figures 6.7, 6.8, and 6.9 show the blocking probabilities, throughputs, and average effective waiting times for different numbers of packet processing tasks,

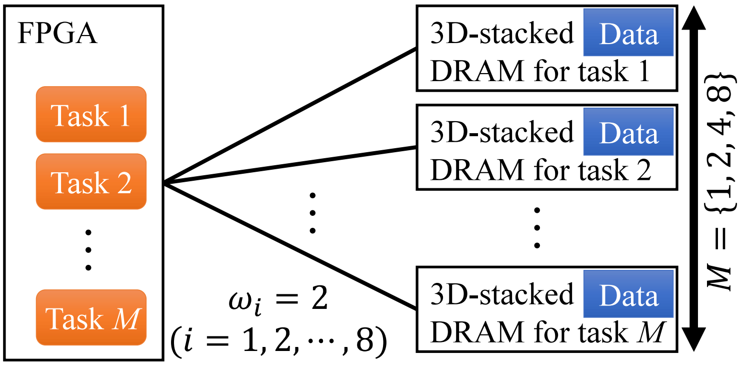


Figure 6.6: Reference architecture for evaluation of dependency on number of packet processing tasks and traffic load.

M, with different architectures, respectively; the set of traffic load ρ is considered as $\{0.1, 0.2, \cdots, 1.9, 2.0\}$. This work denotes the proposed architecture with different M values and the reference architecture with different M values as "*Proposed,* $M = \{1, 2, 4, 8\}$" and "*Ref.,* $M = \{1, 2, 4, 8\}$," respectively.

Figure 6.7 observes that, as ρ increases, the proposed architecture outperforms the reference architecture for each ρ and M, whereas the blocking probability of both architectures increase. Figure 6.8 observes that, in terms of throughput, the proposed architecture slightly outperforms the reference architecture for small ρ, and as ρ increases, the throughput gap between the two architectures becomes large, where the throughput of the reference architecture decreases. This is because, in the reference architecture, memory requests of both first memory accesses and second memory accesses are more likely to be blocked compared to the proposed architecture, as ρ increases. In the proposed architecture, a memory request for the second memory access is issued in the 3D-stacked DRAM just after the first memory access is completed. On the other hand, in the reference architecture, there might be a longer time interval between the first memory access and the second memory access; a memory request for the second memory access is issued after the response of the first memory access is returned to the FPGA, and the issued memory request for the second memory access enters the memory subsystem in the 3D-stacked DRAM after the data transfer from the FPGA to the 3D-stacked

DRAM. As a result, in the reference architecture, the memory request for the second memory access arrives at the memory subsystem after the arrivals of some new memory requests during the time interval from the first memory access, where the queues of the memory subsystem are likely to be full.

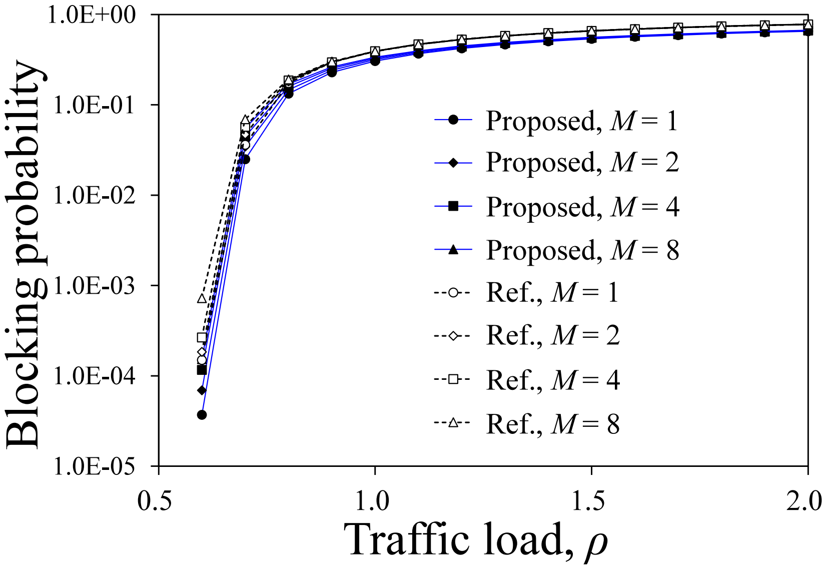


Figure 6.7: Blocking probability depending on number of packet processing tasks and traffic load.

Figure 6.9 observes that the proposed architecture outperforms the reference architecture for each number of packet processing tasks, M. The gap between the proposed architecture and the reference architecture becomes large as M increases. This is because, as M increases, the total number of data transfers between the FPGA and the 3D-stacked DRAMs increases in the reference architecture. In contrast, the proposed architecture has a constant number of data transfers between the FPGA and the 3D-stacked DRAMs when the number of packet processing tasks increases.

Figure 6.9 also observes that the average effective waiting times of the reference architecture with $M > 1$ have peaks when ρ is around 0.7. This is because, in the reference architecture with $M > 1$, packet arrival rates of the second and following packet processing tasks depend on the throughput of the previous packet processing task. Let λ_e^j denotes the throughput of jth packet processing task, where $j \in [1, M]$. Since $\lambda_e^1 \geq \lambda_e^2 \geq \cdots \geq \lambda_e^M$, the

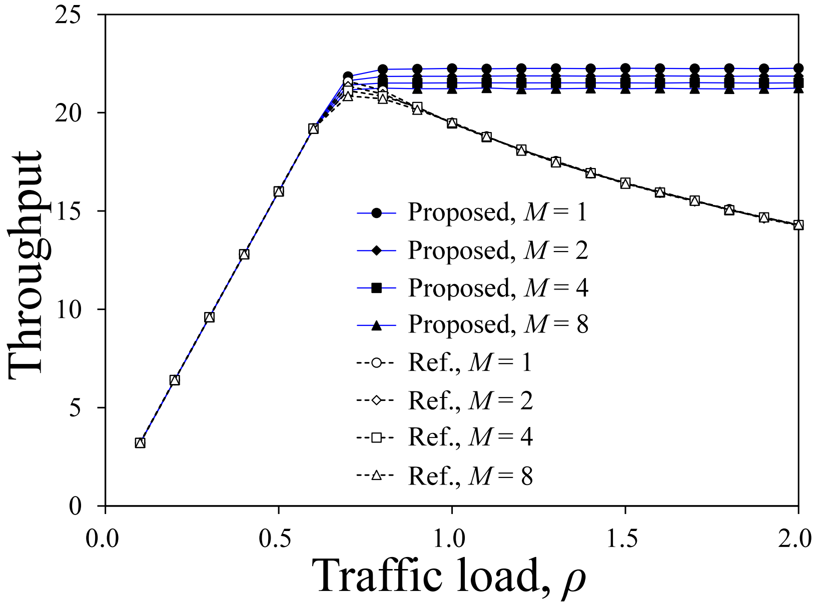


Figure 6.8: Throughput depending on number of packet processing tasks and traffic load.

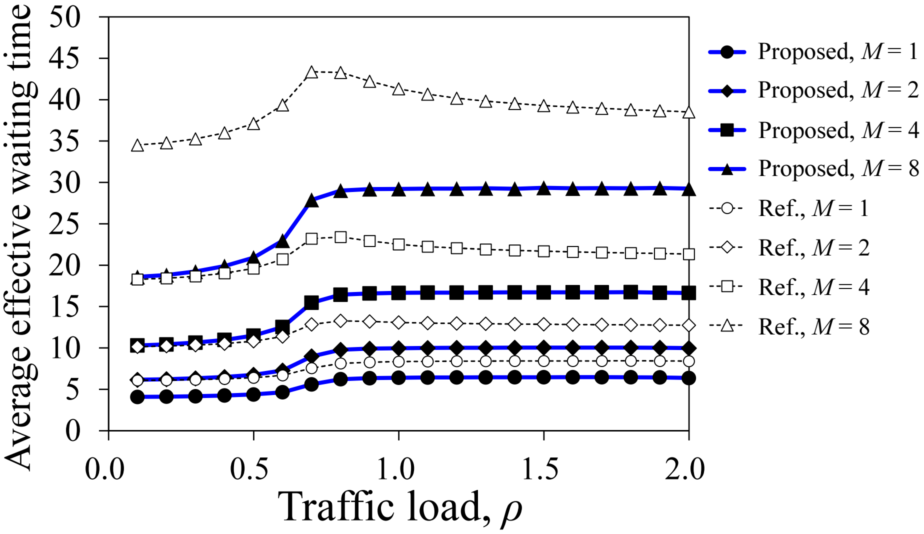


Figure 6.9: Average effective waiting time depending on number of packet processing tasks and traffic load.

throughput decreases when ρ is larger than a certain point, which is observed in Figure 6.8, is supposed to be most significant in the first packet processing task. Consequently, as λ_e^1 decreases when $\rho > 0.7$, the packet arrival rates of the second and following packet processing tasks decrease, which eventually decreases the average effective waiting time. Figure 6.9 also observes that the behavior of the average effective waiting times of the reference architecture with $M > 1$ becomes significant as M increases. This is because, as M increases, the number of packet processing tasks increase, whose average effective waiting time depends on the throughput of the previous task.

6.3.3 Performance dependency on number of vertically cascaded 3D-stacked DRAMs and traffic load

This work considers a memory network with $M = 4$ and $L = \{1, 2, 4, 8\}$ for the packet processing task 2 as depicted in Figure 6.10. Each 3D-stacked DRAM has a packet processing task, whose required numbers of memory accesses are $\omega_1 = \omega_3 = \omega_4 = 2$ and $\omega_2 = 6$.

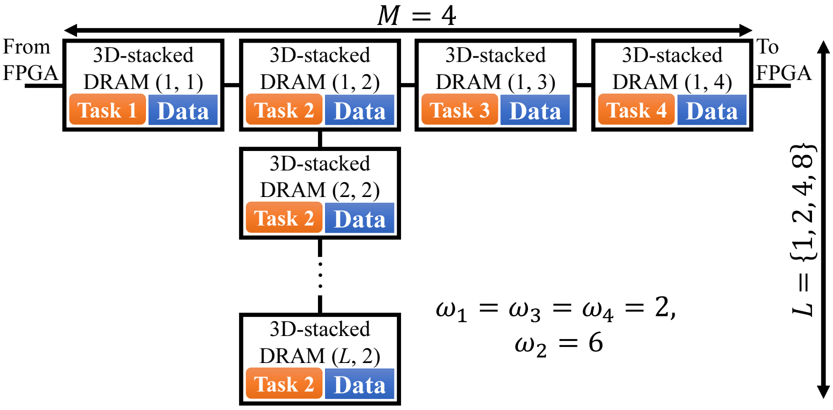


Figure 6.10: Memory network for evaluation of dependency on number of vertically cascaded 3D-stacked DRAMs for task 2 and traffic load.

Figures 6.11, 6.12, and 6.13 show the blocking probabilities, throughputs, and average effective waiting times for different numbers of vertically cascaded 3D-stacked DRAMs for the packet processing task 2, L, respectively; the set of traffic load ρ is considered as $\{0.1, 0.2, \cdots, 1.9, 2.0\}$.

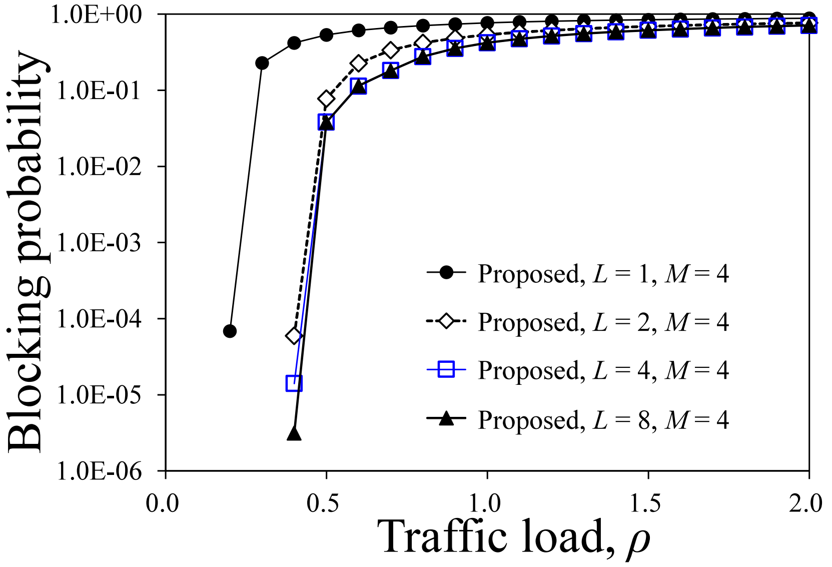


Figure 6.11: Blocking probability depending on number of vertically cascaded 3D-stacked DRAMs and traffic load.

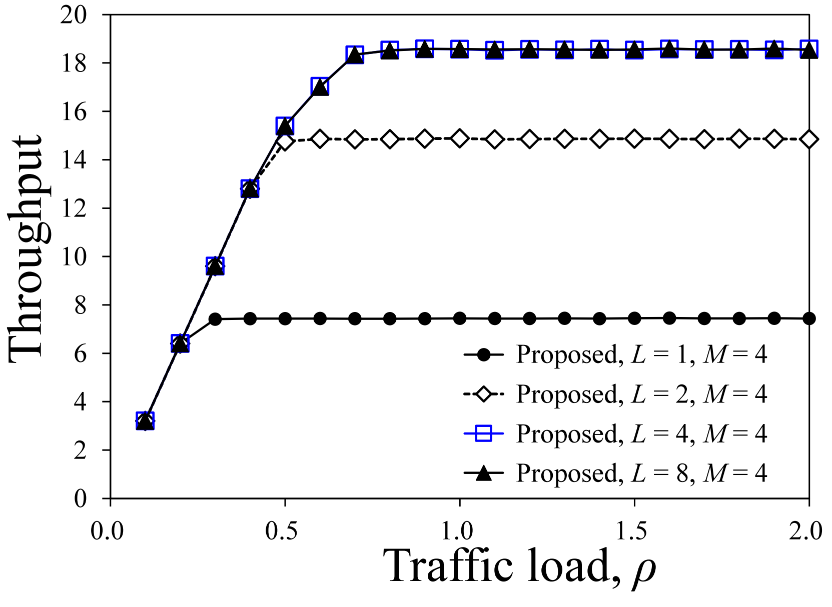


Figure 6.12: Throughput depending on number of vertically cascaded 3D-stacked DRAMs and traffic load.

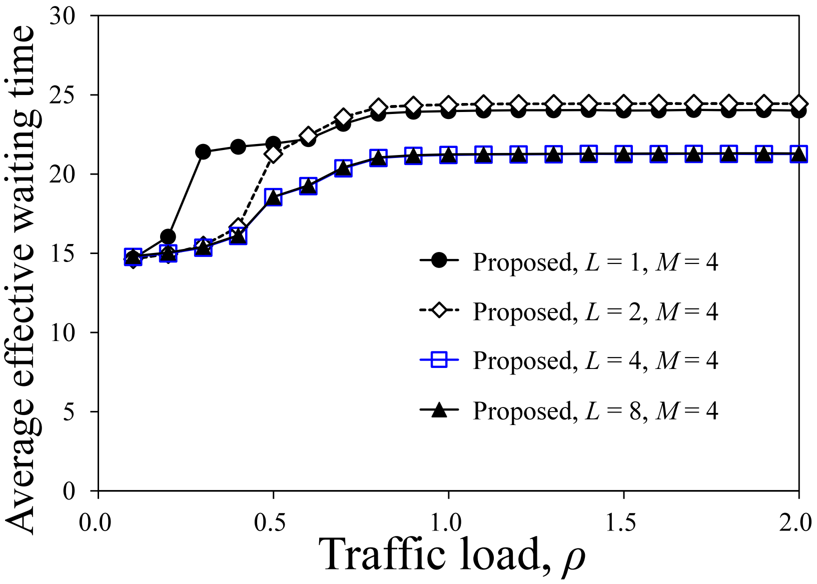


Figure 6.13: Average effective waiting time depending on number of vertically cascaded 3D-stacked DRAMs and traffic load.

Figure 6.11 observes that, as L increases, the blocking probability decreases for each ρ. This is because, as L increases, memory parallelism for the packet processing task 2 increases, which enables the memory network to distribute the traffic load to multiple 3D-stacked DRAMs. Since the required number of memory accesses to complete the packet processing task 2, ω_2, is larger than those of the other tasks, the blocking probability of the 3D-stacked DRAMs for the packet processing task 2 is higher for smaller L, which becomes a bottleneck in the memory network. Additionally, this work observes that, the blocking probability of the proposed architecture with $L = 8$ is almost the same as that with $L = 4$. This result suggests that $L = 4$ 3D-stacked DRAMs are sufficient to perform the packet processing task 2 at the same speed as the other tasks, and more 3D-stacked DRAMs do not increase the performance of the memory network further in this configuration.

Figures 6.12 and 6.13 observe that, as L increases, throughput increases and average effective waiting time decreases. This observation corresponds to the behavior of the blocking probability with respect to L and ρ, where the blocking probability is reduced by distributing the traffic load to L 3D-stacked DRAMs. Additionally, Figure 6.13 observes that, as ρ increases, the average effective waiting time for $L = 1$ increases more rapidly, compared to the cases for $L = \{2, 4, 8\}$. This rapid increase of the average effective waiting time corresponds to the rapid increase of the blocking probability for $L = 1$, as observed in Figure 6.11, where the blocking probability for $L = 1$ is higher than the other condition of L, for smaller ρ.

6.4 Chapter summary

This chapter proposed a parallel memory system architecture that uses a network of 3D-stacked DRAMs to increase throughput and reduce accumulated latency for memory accesses and data transfers between the processors and memories when there are multiple packet processing tasks. In the proposed architecture, packets that enter the memory network receive packet processing tasks at each 3D-stacked DRAM without data transfers between the processors and memories until the packet processing is completed. For a packet processing task that requires more memory accesses to complete the task than the other tasks, the task is allocated to the additional 3D-stacked DRAMs in the memory network to increase memory parallelism. This work evaluated the proposed architecture in terms of the blocking probability, throughput, and average effective waiting time considering different numbers of packet processing tasks and assigned 3D-stacked DRAMs for a particular task, with respect to input traffic load. The evaluation results observed that the proposed architecture reduces the blocking probability by issuing the next memory request inside the 3D-stacked DRAM just after the previous memory access, instead of allowing the arrivals of incoming packets during the data transfers between the memory and processor. Consequently, the proposed architecture increases the throughput and reduces accumulated latency when there are multiple packet processing tasks, compared to the conventional architecture with 3D-stacked DRAM-based parallel memory, where every memory access is accompanied by data transfers. The proposed architecture also reduces the blocking probability and the average effective waiting time by scaling the number of 3D-stacked DRAMs for a particular packet processing task that requires more memory accesses than the other tasks.

Chapter 7

Conclusions

Packet processing requires high memory performance for packet classification, searching, modification, queueing, and so on. Network virtualization is expected to reduce both CAPEX and OPEX by using inexpensive and flexible COTS hardware such as general-purpose computers instead of dedicated network equipment. However, the lack of memory parallelism in general-purpose computers significantly degrades the packet processing performance of VNFs. In particular, large-scale service providers such as telecom operators have depended on conventional dedicated network equipment in order to satisfy the SLA of heavy-duty VNFs that accommodates a large number of subscribers, which prevents network operators from benefiting from network virtualization. This thesis studied four problems regarding parallel memory system architectures for packet processing in network virtualization. Each problem corresponds to the main memory parallelism, integration of the on-chip cache memories of the CPU with the parallel main memory, capacity and parallelism of the on-chip cache memories in the presence of parallel main memory, and accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses, respectively.

Firstly, this thesis proposed a parallel memory system architecture that uses a 3D-stacked memory to increase the memory parallelism of the main memory in general-purpose computers for packet processing in network virtualization. A database searched in packet processing is split into several partial databases, each of which is stored in a memory bank in a memory channel so that the original database comprises all the partial databases in the memory channel. The database in a memory channel is copied to the other memory channels in the 3D-stacked DRAM so that the proposed architecture enhances the memory parallelism by leveraging the memory channel-level parallelism and bank interleaving. According to the packet content, a hash-function-based distributor distributes incoming memory requests into an appropriate memory channel-bank set that contains a partial database in the 3D-stacked DRAM. This work introduced an analytical model of the proposed main memory system architecture that considers the bank interleaving, for two traffic patterns where the arrivals of memory requests are either random or bursty. The analytical results for random arrival of memory requests observed the performance dependency of the proposed architecture on traffic load and the number of bank interleaving. The analytical results for bursty arrival of memory requests observed the performance dependency of the proposed architecture on input traffic burstiness. Based on these analytical results, the calculated packet processing performance, for which IP routing is taken as an example, showed that the proposed architecture increases the packet processing up to 80 Gbps for the smallest-sized IP packet involving both arrival patterns of random and bursty memory requests.

Secondly, this thesis proposed a parallel memory system architecture in general-purpose computers that uses a 3D-stacked memory and on-chip private cache memories to reduce memory access latency in the existence of main memory parallelism for packet processing in network virtualization. The proposed architecture integrates the on-chip private cache memories of each CPU core, that is the L1 and L2 cache memories, with the off-chip 3D-stacked DRAM. Frequently accessed database entries in the 3D-stacked DRAM are copied and stored in the on-chip cache memories. This work explored the memory system architecture in terms of the integration of the on-chip cache memories with the off-chip 3D-stacked DRAM, considering two reference architectures, one with the on-chip L1 and L2 cache memories and the on-chip shared LLC and one without on-chip cache memory. The results observed that the proposed architecture reduces memory access latency by 58 % and 1.8 % and increases throughput by 104 % and 1 % with reducing the blocking probability about 91 % and 18 %, compared to the architecture with the on-chip shared LLC and the architecture without any on-chip cache memory, respectively. The results also observed that the on-chip shared LLC becomes a bottleneck due to the concentration of memory requests, which implies that memory parallelism in on-chip cache memories is inevitable for packet processing in network virtualization when the main memory parallelism coexists.

Thirdly, this thesis proposed a parallel memory system architecture that uses a 3D-stacked memory, on-chip private cache memories, and on-chip LLC slices to increase both capacity and parallelism of the on-chip cache memories in the integration with the parallel main memory for packet processing in network virtualization. The on-chip shared LLC comprises multiple LLC slices, each of which belongs to a CPU core and can be accessed from the other CPU cores. The number of assigned LLC slices to a specific application is configurable. The proposed architecture integrates the LLC slices with the on-chip private cache memories and the off-chip 3D-stacked DRAM. It also has the skip selection logic that determines if a memory request that misses the L2 cache memory skips the LLC based on the statistics of performance counters. The results observed that the proposed architecture reduces memory access latency by 62 % and 12 % and increases throughput by 108 % and 2 % with reducing the blocking probability by 96 % and 50 %, compared to the architectures with the on-chip shared LLC and that without on-chip LLC, respectively. In addition, with a larger number of assigned LLC slices, performance of the proposed architecture increases, while the number of assigned LLC slices does not significantly increase the memory performance if more than a certain number of LLC slices are assigned. Additionally, the results also implied that the benefit of LLC slices decreases as the database size in the system is larger than one point. For such a case, the skip selection logic in the proposed architecture can select skipping the LLC to avoid performance degradation.

Fourthly, this thesis proposed a parallel memory system architecture that uses a network of 3D-stacked memories to increase throughput and reduce accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses. In the proposed architecture, packets that enter the memory network receive packet processing tasks at each 3D-stacked DRAM without data transfers between the processors and memories until the packet processing is completed. For a packet processing task that requires more memory accesses to complete the task than the other tasks, the task is allocated to the additional 3D-stacked DRAMs to increase memory parallelism. This work evaluated the proposed architecture in terms of the blocking probability, throughput, and average effective waiting time considering different numbers of packet processing tasks and assigned 3D-stacked DRAMs for a particular task, with respect to input traffic load. The evaluation results observed that the proposed architecture reduces the blocking probability by issuing the next memory request inside the 3D-stacked DRAM just after the previous memory access, instead of allowing the arrivals of incoming packets during the data transfers between the memory and processor. Consequently, the proposed architecture increases the throughput and reduces accumulated latency when there are multiple packet processing tasks, compared to the architecture with 3D-stacked DRAM-based parallel main memory, where every memory access is accompanied by data transfers. The proposed architecture also reduces the blocking probability and the average effective waiting time by scaling the number of 3D-stacked DRAMs for a particular packet processing task that requires more memory accesses than the other tasks.

The four proposed architectures studied the three fundamental components, which are the main memory, the on-chip private cache memories, and the on-chip LLC slices, respectively, and the memory network, in the parallel memory system of general-purpose computers for packet processing in network virtualization. This thesis provided the parallel memory system architectures to increase VNF performance on top of general-purpose computers by using the 3D-stacked DRAM, the on-chip cache memories, and the distributed placement of the database. These proposed architectures enable the large-scale network operators to apply network virtualization to the burdensome network functions that have large databases, which cannot be the target of network virtualization based on the conventional architectures. This thesis also provided the memory network of 3D-stacked DRAMs for future general-purpose computer architectures, inspired by the memory-centric computing architectures. The memory network architecture eliminates the data transfers between processors and memories during packet processing, which will contribute to the low-latency network and energy-efficient NFVI. The parallelism of the 3D-stacked DRAM-based main memory architecture and the distributed database placement increase the overall memory performance for packet processing. An operator of the computer systems can also select the assignment of parallel LLC slices to an application and whether to skip the LLC depending on the database size and the occurrence of LLC misses to increase memory performance further. The memory network of 3D-stacked DRAMs reduces the blocking probability of multiple memory accesses required for a particular packet processing task, which eventually increases throughput and reduces accumulated latency of data transfers between processors and memories when there are multiple packet processing tasks with memory accesses.

For future works, there can be two directions in packet processing in network virtualization. First, this thesis considers the memory accesses to search the databases in the steady state in which there is no memory write operations. In the network in operation, memory write operations for such as updates of routing tables and classification rules may happen. Therefore, the first direction by expanding the proposed architectures is to consider the memory write operations. Second, this thesis studies parallel memory system architectures in general-purpose computers for packet processing in network virtualization. For packet processing, there are other hardware and software components in general-purpose computers, such as processors, accelerators, virtualization hypervisors, OSes, and programming models to utilize the memory parallelism. Therefore, the second direction is to consider the other components in general-purpose computers for packet processing in network virtualization in addition to the proposed memory system architectures.

Bibliography

[1] R. Giladi, *Network Processors: Architecture, Programming, and Implementation*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[2] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[3] J. Mudigonda, H. M. Vin, and R. Yavatkar, "Overcoming the Memory Wall in Packet Processing: Hammers or Ladders?" in *Proc. 2005 ACM Symp. Arch. Netw. and Commun. Syst. (ANCS)*, 2005, pp. 1–10.

[4] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture optimizations for exploiting memory-level parallelism," in *Proc. 31st IEEE ISCA*, 2004, pp. 76–87.

[5] 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 ISCA*, 2015, pp. 336–348.

[6] 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," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 371–382, Jun. 2009.

[7] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 267–278, Jun. 2009.

[8] 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 HPCA*, 2012, pp. 1–12.

[9] W. Peterson and D. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, Jan. 1961.

[10] *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Std., 1985.

[11] R. Braden, D. Borman, and C. Partridge, "Computing the internet checksum," RFC 1071, Sep. 1988. [Online]. Available: https://tools.ietf.org/html/rfc1071

[12] E. Fredkin, "Trie Memory," *Commun. ACM*, vol. 3, no. 9, pp. 490–499, Sep. 1960.

[13] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. 1998 Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 3, 1998, pp. 1240–1247 vol.3.

[14] 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.

[15] J. Huang and P. Wang, "TCAM-Based IP Address Lookup Using Longest Suffix Split," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 976–989, 2018.

[16] H. Le and V. K. Prasanna, "Scalable Tree-Based Architectures for IPv4/v6 Lookup Using Prefix Partitioning," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1026–1039, 2012.

[17] M. J. Akhbarizadeh, M. Nourani, R. Panigrahy, and S. Sharma, "A TCAM-Based Parallel Architecture for High-Speed Packet Forwarding," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 58–72, 2007.

[18] C. R. Davis, *IPSec: Securing VPNs.* McGraw-Hill Professional, 2001.

[19] R. Oppliger, R. Hauser, and D. Basin, "SSL/TLS session-aware user authentication," *Computer*, vol. 41, no. 3, pp. 59–65, 2008.

[20] Cisco, "Cisco Annual Internet Report (2018-2023)," 2020, Accessed: Jul. 20, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[21] C. Hermsmeyer, H. Song, R. Schlenk, R. Gemelli, and S. Bunse, "Towards 100G Packet Processing: Challenges and Technologies," *Bell Labs Technical Journal*, vol. 14, no. 2, pp. 57–79, 2009.

[22] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A Survey on 5G Usage Scenarios and Traffic Models," *IEEE Commun. Surv. Tut.*, vol. 22, no. 2, pp. 905–929, 2020.

[23] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things J.*, vol. 4, no. 5, pp. 1125–1142, 2017.

[24] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[25] ETSI. (2014, Sep) Mobile-Edge Computing – Introductory Technical White Paper. Accessed: Aug. 20, 2020. [Online]. Available: https://portal.etsi.org

[26] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[27] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[28] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, “Mobile-Edge Computing Architecture: The role of MEC in the Internet of Things,” *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 84–91, 2016.

[29] L. Le, D. Sinh, B. P. Lin, and L. Tung, “Applying Big Data, Machine Learning, and SDN/NFV to 5G Traffic Clustering, Forecasting, and Management,” in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, 2018, pp. 168–176.

[30] ETSI, “Network Functions Virtualisation – Introductory White Paper,” Oct. 2012, Accessed: Aug. 20, 2020. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[31] Broadcom, “Engineered Elephant Flows in Large Scale CLOS Networks,” Mar. 2014, Accessed: Aug. 20, 2020. [Online]. Available: https://docs.broadcom.com/doc/1211168569445

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[33] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow, and G. Parulkar, “ONOS: Towards an Open, Distributed SDN OS,” in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Chicago, Illinois, USA, 2014, pp. 1–6.

[34] Faucet: Open source SDN Controller for production networks. Accessed: Aug. 20, 2020. [Online]. Available: https://faucet.nz/

[35] Open vSwitch. Accessed: Aug. 20, 2020. [Online]. Available: http://www.openvswitch.org/

[36] Lagopus switch and router. Accessed: Aug. 20, 2020. [Online]. Available: http://www.lagopus.org/

[37] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” RFC 6241, Jun. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6241

[38] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, Oct. 2010. [Online]. Available: https://tools.ietf.org/html/rfc6020

[39] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[40] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://tools.ietf.org/html/rfc6020

[41] What is VPP? Accessed: Aug. 20, 2020. [Online]. Available: https://wiki.fd.io

[42] Segment Routing - P4. Accessed: Aug. 20, 2020. [Online]. Available: https://www.segment-routing.net/open-software/P4/

[43] ETSI, "Network Functions Virtualisation – Update White Paper," Oct. 2013. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[44] M.-P. Odini, "OpenSource MANO," *IEEE Softwarization*, 2016, Accessed: Sep. 20, 2020. [Online]. Available: https://sdn.ieee.org/newsletter/july-2016/opensource-mano

[45] G. A. Carella and T. Magedanz, "Open Baton: A Framework for Virtual Network Function Management and Orchestration for Emerging Software-Based 5G Networks," *IEEE Softwarization*, 2016, Accessed: Sep. 20, 2020. [Online]. Available: https://sdn.ieee.org/newsletter/july-2016/open-baton

[46] OpenStack: Open Source Cloud Computing Infrastructure. Accessed: Aug. 20, 2020. [Online]. Available: https://www.openstack.org/

[47] I. Takouna, W. Dawoud, and C. Meinel, "Analysis and simulation of hpc applications in virtualized data centers," in *2012 IEEE Int. Conf. Green Comput. and Commun.*, 2012, pp. 498–507.

[48] Z. Cui, L. Xia, P. G. Bridges, P. A. Dinda, and J. R. Lange, "Optimizing overlay-based virtual networking through optimistic interrupts and cut-through forwarding," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage and Anal. (SC)*, 2012, pp. 1–11.

[49] J. Camacho and J. Flich, "HPC-mesh: A homogeneous parallel concentrated mesh for fault-tolerance and energy savings," in *Proc. ACM/IEEE 7th Symp. Arch. Netw. and Commun. Syst. (ANCS)*, 2011, pp. 69–80.

[50] Cisco, "The Cisco Flow Processor: Cisco's Next Generation Network Processor Solution Overview," Accessed: Aug. 20, 2020. [Online]. Available: https://www.cisco.com

[51] B. Wheeler, "TOMAHAWK 4 SWITCH FIRST TO 25.6 TBPS," The Linley Group, Tech. Rep., Dec. 2019.

[52] Cisco, "Cisco Nexus 3000 Switch Architecture," Accessed: Aug. 20, 2020. [Online]. Available: https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2018/pdf/BRKDCN-3734.pdf

[53] 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.

[54] K. Yamazaki, Y. Ukon, S. Yoshida, S. Hatta, Y. Sekihara, S. Ohteru, T. Kawamura, T. Hatano, K. Nitta, and A. Miyazaki, "Flow Cache Cleansing with FPGA Hash Pipe for Highly Stabilized Software Data Plane," in *Proc. IEEE 19th Int. Conf. High Perform. Switching and Routing (HPSR)*, 2018, pp. 1–6.

[55] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. New York, NY, USA: Elsevier, 2012.

[56] A. Kotabe, K. Osada, N. Kitai, M. Fujioka, S. Kamohara, M. Moniwa, S. Morita, and Y. Saitoh, "A low-power four-transistor SRAM cell with a stacked vertical poly-silicon PMOS and a dual-word-voltage scheme," *IEEE J. Solid-State Circuits*, vol. 40, no. 4, pp. 870–876, 2005.

[57] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, “A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram,” in *Proc. 2005 IEEE Int. Elect. Devices Meeting (IEDM)*, 2005, pp. 459–462.

[58] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. Chow, A. Cabrini, G. M. A. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande, “A Multi-Level-Cell Bipolar-Selected Phase-Change Memory,” in *Proc. 2008 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2008, pp. 428–625.

[59] F. Zahoor, T. Z. A. Zulkifli, and F. A. Khanday, “Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications,” *Nanoscale Res. Lett.*, vol. 15, no. 90, 2020.

[60] D. Takashima, “Overview of FeRAMs: Trends and perspectives,” in *Proc. 2011 11th Annu. Non-Volatile Memory Technol. Symp. (NVMT)*, 2011, pp. 1–6.

[61] B. Gervasi, “Will Carbon Nanotube Memory Replace DRAM?” *IEEE Micro*, vol. 39, no. 2, pp. 45–51, 2019.

[62] M. Komalan, O. H. Rock, M. Hartmann, S. Sakhare, C. Tenllado, J. I. Gómez, G. S. Kar, A. Furnemont, F. Catthoor, S. Senni, D. Novo, A. Gamatie, and L. Torres, “Main memory organization trade-offs with DRAM and STT-MRAM options based on gem5-NVMain simulation frameworks,” in *Proc. 2018 Design, Automat. Test in Eur. Conf. Exhib. (DATE)*, 2018, pp. 103–108.

[63] Y. Jin, M. Shihab, and M. Jung, “Area, Power, and Latency Considerations of STT-MRAM to Substitute for Main Memory,” in *Proc. The Memory Forum, co-located with the 41st Int. Symp. Comput. Arch. (ISCA)*, 2014.

[64] M. Qiu, Z. Chen, Z. Ming, X. Qin, and J. Niu, "Energy-Aware Data Allocation With Hybrid Memory for Mobile Cloud Systems," *IEEE Syst. J.*, vol. 11, no. 2, pp. 813–822, 2017.

[65] K. Cai and K. A. S. Immink, "Cascaded channel model, analysis, and hybrid decoding for spin-torque transfer magnetic random access memory," *IEEE Trans. Magn.*, vol. 53, no. 11, pp. 1–11, 2017.

[66] M.-T. Chang, S.-L. Lu, and B. Jacob, "Impact of Cache Coherence Protocols on the Power Consumption of STT-RAM-Based LLC," in *Proc. The Memory Forum, co-located with 41st Int. Symp. Comput. Arch. (ISCA)*, 2014.

[67] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[68] F. Zane, Girija Narlikar, and A. Basu, "Coolcams: power-efficient TCAMs for forwarding engines," in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 1, 2003, pp. 42–52.

[69] THE NEW INTEL XEON SCALABLE PROCESSOR (FORMERLY SKYLAKE-SP). Accessed: Aug. 20, 2020. [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

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

[71] New Intel Mesh Architecture: The 'Superhighway' of the Data Center. Accessed: Aug. 20, 2020. [Online]. Available: https://silix.com.br/pdf/Intel/Intel_Mesh_Whitepaper.pdf?0f3648&0f3648

[72] 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, 2014.

[73] 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, 2013.

[74] A. Farshin, A. Roozbeh, G. Q. Maguire, and D. Kostić, "Make the Most out of Last Level Cache in Intel Processors," in *Proc. 14th Eur. Conf. Comput. Syst. (EuroSys)*, no. 8, Mar. 2019, pp. 1–17.

[75] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Systematic Reverse Engineering of Cache Slice Selection in Intel Processors," in *Proc. 2015 IEEE Euromicro Conf. Digit. Syst. Design (DSD)*, 2015, pp. 629–636.

[76] Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. Accessed: Aug. 20, 2020. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[77] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Proc. 2012 Symp. VLSI Technol. (VLSIT)*, 2012, pp. 87–88.

[78] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. 2011 IEEE Symp. High Perform. Chips (Hot Chips)*, 2011, pp. 1–24.

[79] H. Jun, J. Cho, K. Lee, H. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in *Proc. 2017 IEEE Int. Memory Workshop (IMW)*, 2017, pp. 1–4.

[80] J. Macri, "AMD's next generation GPU and high bandwidth memory architecture: FURY," in *Proc. 2015 IEEE Symp. High Perform. Chips (Hot Chips)*, 2015, pp. 1–26.

[81] "Hybrid Memory Cube Specification 2.1." Accessed: Aug. 20, 2020. [Online]. Available: http://hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf

[82] M. A. Alves, M. Diener, P. C. Santos, and L. Carro, "Large vector extensions inside the HMC," in *Proc. 2016 Design, Automat. Test in Eur. Conf. Exhib. (DATE)*, 2016, pp. 1249–1254.

[83] D. Jeon, K. Park, and K. Chung, "HMC-MAC: Processing-in Memory Architecture for Multiply-Accumulate Operations with Hybrid Memory Cube," *IEEE Comput. Arch. Lett.*, vol. 17, no. 1, pp. 5–8, 2018.

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

[85] 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)*, 2013, pp. 185–192.

[86] Y. Eckert, N. Jayasena, and G. Loh, "Thermal Feasibility of Die-Stacked Processing in Memory," in *Proc. 2nd Workshop on Near-Data Processing (WoNDP), in conjunction with 47th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2014, pp. 1–5.

[87] 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.

[88] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. 39th Annu. Int. Symp. Comput. Arch. (ISCA)*, 2012, pp. 1–12.

[89] R. S. Williams, "What's Next? [The end of Moore's law]," *Comput. Sci. Eng*, vol. 19, no. 2, pp. 7–13, 2017.

[90] E. P. DeBenedictis, "It's Time to Redefine Moore's Law Again," *Computer*, vol. 50, no. 2, pp. 72–75, 2017.

[91] D. B. Ingerly, S. Amin, L. Aryasomayajula, A. Balankutty, D. Borst, A. Chandra, K. Cheemalapati, C. S. Cook, R. Criss, K. Enamul, W. Gomes, D. Jones, K. C. Kolluru, A. Kandas, G. . Kim, H. Ma,

D. Pantuso, C. F. Petersburg, M. Phen-givoni, A. M. Pillai, A. Sairam, P. Shekhar, P. Sinha, P. Stover, A. Telang, and Z. Zell, “Foveros: 3D Integration and the use of Face-to-Face Chip Stacking for Logic Devices,” in *2019 IEEE Int. Electron Devices Meeting (IEDM)*, 2019, pp. 19.6.1–19.6.4.

[92] C. Prasad, S. Chugh, H. Greve, I. Ho, E. Kabir, C. Lin, M. Maksud, S. R. Novak, B. Orr, K. W. Park, A. Schmitz, Z. Zhang, P. Bai, D. B. Ingerly, E. Armagan, H. Wu, P. Stover, L. Hibbeler, M. O’Day, and D. Pantuso, “Silicon Reliability Characterization of Intel’s Foveros 3D Integration Technology for Logic-on-Logic Die Stacking,” in *2020 IEEE Int. Rel. Phys. Symp. (IRPS)*, 2020, pp. 1–5.

[93] Intel Data Plane Development Kit. Accessed: Aug. 20, 2020. [Online]. Available: https://www.dpdk.org/

[94] PCI-SIG Single Root I/O Virtualization (SR-IOV) Support in Intel Virtualization Technology for Connectivity. Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com

[95] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click Modular Router,” *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[96] K. Fall, G. Iannaccone, M. Manesh, S. Ratnasamy, K. Argyraki, M. Dobrescu, and N. Egi, “RouteBricks: Enabling General Purpose Network Infrastructure,” *SIGOPS Oper. Syst. Rev.*, vol. 45, no. 1, pp. 112–125, Feb. 2011.

[97] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, “High-Speed Software Data Plane via Vectorized Packet Processing,” *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 97–103, 2018.

[98] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen, “Scalable, High Performance Ethernet Forwarding with CuckooSwitch,” in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 97–108.

[99] M. Bando and H. J. Chao, "FlashTrie: Hash-based Prefix-Compressed Trie for IP Route Lookup Beyond 100Gbps," in *Proc. 2010 IEEE INFOCOM*, 2010, pp. 1–9.

[100] J. Huang, R. R. Puli, P. Majumder, S. Kim, R. Boyapati, K. H. Yum, and E. J. Kim, "Active-routing: Compute on the way for near-data processing," in *Proc. IEEE Int. Symp. High Perform. Comput. Arch. (HPCA)*, 2019, pp. 674–686.

[101] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The mondrian data engine," *ACM SIGARCH Comput. Arch. News*, vol. 45, no. 2, pp. 639–651, 2017.

[102] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, "Scalable high-performance elastic software OpenFlow switch in userspace for wide-area network," in *Proc. Open Netw. Summit (ONS)*, 2014.

[103] R. Rahimi, M. Veeraraghavan, Y. Nakajima, H. Takahashi, Y. Nakajima, S. Okamoto, and N. Yamanaka, "A high-performance OpenFlow software switch," in *Proc. IEEE 17th Int. Conf. High Perform. Switching Routing (HPSR)*, 2016, pp. 93–99.

[104] Intel 64 and IA-32 Architectures Software Developer's Manual. Accessed: Aug. 20, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html

[105] G. Lettieri, V. Maffione, and L. Rizzo, "A survey of fast packet I/O technologies for network function virtualization," in *Proc. Int. Conf. High Perfor. Comput. (HiPC)*, 2017, pp. 579–590.

[106] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *Proc. 2015 ACM/IEEE Symp. Arch. Netw. Commun. Syst. (ANCS)*, May 2015, pp. 5–16.

[107] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma, "Towards optimal adaptation of nfv packet processing to modern cpu memory ar-

chitectures," in *Proc. 2nd Workshop Cloud-Assisted Netw.*, 2017, pp. 7–12.

[108] 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. 107 161–107 170, 2019.

[109] 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 (MICRO)*, 2014, pp. 25–37.

[110] 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, 2015.

[111] 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)*, 2018, pp. 559–572.

[112] 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. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, 2017, pp. 187–197.

[113] Y. Yu and N. K. Jha, "Energy-Efficient Monolithic Three-Dimensional On-Chip Memory Architectures," *IEEE Trans. Nanotechnol.*, vol. 17, no. 4, pp. 620–633, 2018.

[114] L. B. Kish, "End of Moore's law: thermal (noise) death of integration in micro and nano electronics," *Phys. Lett. A*, vol. 305, no. 3, pp. 144–149, Dec. 2002.

[115] T. N. Theis and H.-S. P. Wong, "The End of Moore's Law: A New Beginning for Information Technology," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 41–50, Mar. 2017.

[116] “Disaggregated Servers Drive Data Center Efficiency and Innovation,” Jun. 2017, Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.co.jp/content/www/jp/ja/it-management/intel-it-best-practices/disaggregated-server-architecture-drives-data-center-efficiency-paper.html

[117] A. Roozbeh, J. Soares, G. Q. Maguire, F. Wuhib, C. Padala, M. Mahloo, D. Turull, V. Yadhav, and D. Kostić, “Software-Defined “Hardware” Infrastructures: A Survey on Enabling Technologies and Open Research Directions,” *IEEE Commun. Surv. Tut.*, vol. 20, no. 3, pp. 2454–2485, 2018.

[118] H. A. D. Nguyen, J. Yu, M. A. Lebdeh, M. Taouil, S. Hamdioui, and F. Catthoor, “A Classification of Memory-Centric Computing,” *J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 13, Jan. 2020.

[119] K. Kim, S. Shin, and S.-M. Kang, “Stateful logic pipeline architecture,” in *Proc. 2011 IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2011, pp. 2497–2500.

[120] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, “MAGIC—Memristor-aided logic,” *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[121] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, “ReVAMP: ReRAM based VLIW architecture for in-memory computing,” in *Proc. Design, Automat. Test in Eur. Conf. Exhib. (DATE)*, 2017, pp. 782–787.

[122] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, “Drisa: A dram-based reconfigurable in-situ accelerator,” in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2017, pp. 288–301.

[123] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, “Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories,” in *Proc. 53nd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, 2016, pp. 1–6.

[124] D. Fujiki, S. Mahlke, and R. Das, “In-memory data parallel processor,” *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[125] L. Eeckhout, “Computer architecture performance evaluation methods,” *Synthesis Lectures Comput. Arch.*, vol. 5, no. 1, pp. 1–145, 2010.

[126] A. Akram and L. Sawalha, “A Survey of Computer Architecture Simulation Techniques and Tools,” *IEEE Access*, vol. 7, pp. 78 120–78 145, 2019.

[127] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The Gem5 Simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[128] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation,” in *Proc. 2011 Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2011, pp. 1–12.

[129] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, “McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling,” in *Proc. 2013 IEEE Int. Symp. Perf. Anal. Syst. Software (ISPASS)*, 2013, pp. 74–85.

[130] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, “The M5 simulator: Modeling networked systems,” *IEEE MICRO*, vol. 26, no. 4, pp. 52–60, 2006.

[131] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset,” *ACM SIGARCH Comput. Arch. News*, vol. 33, no. 4, pp. 92–99, 2005.

[132] D. Genbrugge, S. Eyerman, and L. Eeckhout, “Interval simulation: Raising the level of abstraction in architectural simulation,” in *Proc. 16th Int. Symp. High-Perform. Comput. Arch. (HPCA)*, 2010, pp. 1–12.

[133] 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. 75 500–75 514, 2019.

[134] 2nd Generation Intel Xeon Scalable Processors Brief. Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/processors/xeon/2nd-gen-xeon-scalable-processors-brief.html

[135] Hybrid Memory Cube Controller IP Core User Guide. Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/nik1412377950681.html

[136] Xilinx HMC Controller PG216 - XHMC v1.0 Product Guide (v1.0). Accessed: Aug. 20, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xhmc/v1_0/pg216-xhmc.pdf

[137] High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide. Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-20031.pdf

[138] HIGH BANDWIDTH MEMORY (HBM2) CONTROLLER AND PHY. Accessed: Aug. 20, 2020. [Online]. Available: http://www.open-silicon.com/high-bandwidth-memory-ip/

[139] AXI High Bandwidth Memory Controller v1.0. Accessed: Aug. 20, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf

[140] Intel STRATIX 10 MX FPGAS. Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/sip/stratix-10-mx.html

[141] 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. 2011 Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2011, pp. 80–85.

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

[143] 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. 2018 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 107–116.

[144] K. Leung, V. O. k. Li, and D. Yang, "An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 522–535, 2007.

[145] S. Govind, R. Govindarajan, and J. Kuri, "Packet Reordering in Network Processors," in *Proc. 2007 IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2007, pp. 1–10.

[146] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling Internet Routers Using Optics," in *Proc. 2003 Conf. Appl., Technol., Arch., Protocols Comput. Commun. (SIGCOMM)*, 2003, pp. 189–200.

[147] Y. Wu and G. Nong, "scalable pipeline architecture for IPv4/IPv6 route lookup," in *Proc. 18th IEEE Int. Conf. Netw. (ICON)*, 2012, pp. 416–421.

[148] H. Le and V. K. Prasanna, "Scalable Tree-Based Architectures for IPv4/v6 Lookup Using Prefix Partitioning," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1026–1039, 2012.

[149] J. D. C. Little, "A Proof for the Queuing Formula: $L = \lambda W$," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, Jun. 1961.

[150] C. Milne, "Transient Behaviour of the Interrupted Poisson Process," *J. Roy. Statist. Soc. Ser. B (Methodological)*, vol. 44, no. 3, pp. 398–405, 1982.

[151] Q. Ren and H. Kobayashi, "Transient solutions for the buffer behavior in statistical multiplexing," *Perform. Eval.*, vol. 23, no. 1, pp. 65–87, 1995.

[152] A. Reibman and K. Trivedi, "Numerical transient analysis of markov models," *Comput. Oper. Res.*, vol. 15, no. 1, pp. 19–36, 1988.

[153] C. Bandi, D. Bertsimas, and N. Youssef, "Robust transient analysis of multi-server queueing systems and feed-forward networks," *Queueing Syst.*, vol. 89, no. 3, pp. 351–413, Aug. 2018.

[154] B. Liu, Q. Xie, and E. Modiano, "Reinforcement Learning for Optimal Control of Queueing Systems," 2019.

[155] H. Yao, X. Yuan, P. Zhang, J. Wang, C. Jiang, and M. Guizani, "Machine Learning Aided Load Balance Routing Scheme Considering Queue Utilization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7987–7999, 2019.

[156] 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. 2017 IEEE Int. Symp. Workload Characterization (IISWC)*, 2017, pp. 66–75.

[157] 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. 2018 IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2018, pp. 99–108.

[158] T.Korikawa, A. Kawabata, F. He, and E. Oki, "Packet Processing Architecture with Off-Chip Last Level Cache Using Interleaved 3D-Stacked DRAM," *IEICE Trans. Commun.*, vol. E104-B, no. 2, pp. 149–157, Feb. 2021.

[159] Intel Product Brief "Enabling UPI and PCIe Gen4 Accelerators". Accessed: Aug. 20, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/stratix-10-dx-product-brief.pdf

[160] E. Cohen and H. Kaplan, "Proactive caching of DNS records: addressing a performance bottleneck," in *Proc. 2001 Symp. Appl. Internet (SAINT)*, 2001, pp. 85–94.

[161] T. Chiueh and P. Pradhan, "High-performance IP routing table lookup using CPU caching," in *Proc. 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 3, 1999, pp. 1421–1428.

[162] S. Ihm and V. S. Pai, "Towards Understanding Modern Web Traffic," in *Proc. 2011 ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2011, pp. 295–312.

[163] Jaeyeon 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, 2002.

[164] Chen Wang, Li Xiao, Yunhao Liu, and Pei Zheng, "DiCAS: An Efficient Distributed Caching Mechanism for P2P Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 10, pp. 1097–1109, 2006.

[165] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," in *Proc. 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 1, 1999, pp. 126–134 vol.1.

[166] 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, March 2012.

[167] Intel 64 and IA-32 Architectures Optimization Reference Manual. Accessed: Aug. 20, 2020. [Online]. Available:

https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-optimization-reference-manual.html

[168] T. Korikawa, A. Kawabata, F. He, and E. Oki, "Packet Processing Architecture Using Last-Level-Cache Slices and Interleaved 3D-Stacked DRAM," *IEEE Access*, vol. 8, pp. 59 290–59 304, 2020.

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

[170] 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. 20 170 324–20 170 324, 2017.

[171] Y. Tian, S. Puthoor, J. L. Greathouse, B. M. Beckmann, and D. A. Jiménez, "Adaptive GPU Cache Bypassing," in *Proc. 8th Workshop General Purpose Process. Using GPUs (GPGPU)*, 2015, pp. 25–35.

[172] M. Kharbutli and Y. Solihin, "Counter-Based Cache Replacement and Bypassing Algorithms," *IEEE Trans, Comput.*, vol. 57, no. 4, pp. 433–447, 2008.

[173] S. Gupta, H. Gao, and H. Zhou, "Adaptive Cache Bypassing for Inclusive Last Level Caches," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process. (IPDPS)*, 2013, pp. 1243–1253.

[174] 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., IEEE 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst. (HPCC-CSS-ICESS)*, 2015, pp. 972–976.

[175] 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, 2016.

[176] S. Mittal, “A Survey of Cache Bypassing Techniques,” *J. Low Power Electron. Appl.*, vol. 6, no. 2, 2016.

[177] T. Korikawa and E. Oki, “Memory Network Architecture for Packet Processing in Functions Virtualization,” submitted to IEEE conference (under review).

[178] G. Kim, J. Kim, J. H. Ahn, and J. Kim, “Memory-centric system interconnect design with Hybrid Memory Cubes,” in *Proc. 22nd Int. Conf. Parallel Arch. Compilation Techn.*, 2013, pp. 145–155.

[179] P. Rosenfeld, “Performance Exploration of the Hybrid Memory Cube,” Ph.D. dissertation, University of Maryland, 2014.

Publication List

Journal Papers

1. **T. Korikawa**, A. Kawabata, F. He, and E. Oki, “Packet Processing Architecture with Off-Chip Last Level Cache Using Interleaved 3D-Stacked DRAM,” *IEICE Transactions on Communications*, vol. E104-B, No. 2, pp. 149–157, 2021.

2. **T. Korikawa**, A. Kawabata, F. He, and E. Oki, “Packet Processing Architecture Using Last-Level-Cache Slices and Interleaved 3D-Stacked DRAM,” *IEEE Access*, vol. 8, pp. 59290–59304, 2020.

3. **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.

International Conference Papers

1. **T. Korikawa**, A. Kawabata, F. He, and E. Oki, “Packet Processing Architecture With Off-Chip LLC Using Interleaved 3D-Stacked DRAM,” in *Proceedings of IEEE International Conference on High Performance Switching and Routing (HPSR)*, Xi’An, China, 2019, pp. 1–6.

2. **T. Korikawa**, A. Kawabata, F. He, and E. Oki, “Carrier-Scale Packet Processing System Using Interleaved 3D-Stacked DRAM,” in *Proceedings of IEEE International Conference on Communications (ICC)*, Kansas City, MO, 2018, pp. 1–6.