

GCCache: A Non-blocking and Low-latency General-purpose Context Cache Design for RDMA NICs

Zhixiang Zhao* **, Zhichuan Guo* **, Mangu Song*, Zhiyuan Ling*

*National Network New Media Engineering Research Center, Institute of Acoustics,
Chinese Academy of Sciences, Beijing 100190, China

**School of Electronic, Electrical and Communication Engineering,
University of Chinese Academy of Sciences, Beijing 100049, China

zhaozx@dsp.ac.cn, guozc@dsp.ac.cn, songmg@dsp.ac.cn, lingzy@dsp.ac.cn

Abstract—Remote Direct Memory Access (RDMA) network interface cards (NICs) are widely used in AI large-scale model training and intelligent computing centers. However, current NIC caches face several challenges, such as head-of-line (HOL) blocking, repeated processing of misses to the same memory address, high latency, and poor compatibility. To address these challenges, for the first time we propose GCCache: a non-blocking, low-latency, high-performance general-purpose cache architecture for NICs. GCCache decouples hit processing and miss processing to enable non-blocking cache access and resolve HOL blocking. In addition, an outstanding request management mechanism ensures that for multiple misses to the same memory address, all corresponding requests are served immediately once the context is fetched, thereby saving PCIe bandwidth. Moreover, multi-request parallel processing and pipeline design reduce access latency and support context prefetching. GCCache supports all types of RDMA contexts (QPC, MPT, MTT, etc.) and multi-level page table queries. To the best of our knowledge, GCCache outperforms all currently published RDMA cache architectures in terms of latency and on-chip storage resources, with a cache processing latency of only 5 clock cycles and a throughput of 100 Gbps.

Index Terms—RDMA, cache architecture, FPGA.

I. INTRODUCTION

RDMA plays a critical role in AI clusters [1], data center networks (DCNs) [2], and high-performance computing (HPC) environments [3] with its low latency and high throughput characteristics. Due to the hardware offloading nature of NICs, on-chip resources are insufficient to accommodate a large number of connections. Under normal operation, the NIC retrieves context entries directly from the cache for low-latency access, and accesses the slower host memory only upon a cache miss. Therefore, the performance of the NIC context cache is critical to overall system efficiency [4] [5] [6] [7].

Current designs of NIC context caches faces several critical challenges:

- **Head-of-Line Blocking:** Most existing NIC caches are designed using the principle of first request first response [7]. If the first request misses the cache, it needs to fetch the context from the host, even if the subsequent request

can hit the cache, it has to wait until the previous request gets a response before accessing the cache.

- **Redundant Host Accesses:** Consecutive requests received by the cache may be identical, if the first request misses the cache, the subsequent request redundant accesses waste PCIe bandwidth. Existing commercial NICs and academic studies have not mentioned any solutions to this problem.
- **High Latency:** When the NIC data processing pipeline queries the cache, it must wait for the response. High cache access latency directly degrades system performance [4]. At a frequency of 200 MHz, if the NIC forwards 1024-byte packets at 100 Gbps, the cache response must be returned within 16 clock cycles.
- **Lack of Compatibility:** An NIC usually has different types of contexts (QPC, CQC, SRQC, MPT, MTT), and it takes a long development cycle to design and develop specialized caches for different types of contexts separately.

To address these issues, we propose GCCache, a non-blocking, low-latency, high-performance general-purpose cache architecture for NICs. GCCache makes the following three key contributions:

- 1) GCCache separates hit processing and miss handling to eliminate HOL blocking and achieve high performance and low latency. GCCache outperforms all currently published 100 Gbps RDMA cache architecture in terms of latency and on-chip storage resources.
- 2) GCCache prevents duplicate host memory accesses for identical miss requests and ensures that all such requests are served immediately once the context is returned.
- 3) GCCache is compatible with all types of RDMA contexts (QPC, CQC, SRQC, MPT, MTT).

II. BACKGROUND AND RELATED WORKS

A. RDMA Basics

The typical architecture and communication model of the NIC are shown in Fig. 1. The basic unit for maintaining

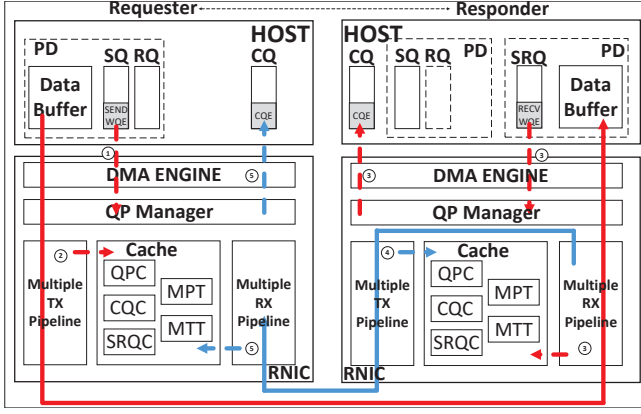


Fig. 1: Typical RDMA architecture and communication conceptual model, illustrated with a two-sided operation example.

connections in RDMA is the Queue Pair (QP), which consists of a Send Queue (SQ) and a Receive Queue (RQ) used to store Work Queue Elements (WQEs). The data buffers associated with WQEs reside in user-pre-registered Memory Regions (MRs), and the Protection Domain (PD) of an MR must match the PD of the QP that stores the corresponding WQE. When the task is completed, the RNIC writes a Completion Queue Element (CQE) into the Completion Queue (CQ) to notify the software. To conserve resources on the receiver side, the InfiniBand (IB) specification also defines a Shared Receive Queue (SRQ). When the software creates a QP, CQ, or SRQ, it simultaneously creates the corresponding QP Context (QPC), CQ Context (CQC), and SRQ Context (SRQC) to maintain their states.

Since RDMA enables the NIC to directly access host memory via physical addresses, the software must establish a Memory Translation Table (MTT), which the RNIC uses to translate virtual addresses into physical addresses. To prevent unauthorized memory access, the software also stores each MR's PD number, access permissions, base address, and size in the Memory Protection Table (MPT). The RNIC verifies access rights before accessing the MR. Typically, all these contexts (QPC, CQC, SRQC, MPT, and MTT) are stored in the host, while only a subset is cached in RNIC.

Taking the RDMA SEND two-sided operation as an example, the detailed process is as follows:

- 1) The requester host rings the doorbell to notify the requester RNIC to fetch a WQE (step 1).
- 2) The requester RNIC queries the cache and sends the data to the responder (step 2).
- 3) The responder RNIC queries the cache, fetches the WQE from the RQ (or SRQ), writes the data into responder data buffer, and sends a CQE into the CQ (step 3).
- 4) The responder RNIC queries the cache, returns an ACK to the requester RNIC (step 4).
- 5) The requester RNIC queries the cache, writes a CQE into the CQ. (step 5).

B. Related Works

When the number of context entries exceeds the capacity of the hardware cache, the RNIC experiences cache misses and must access host memory, leading to a significant degradation in performance [8] [9]. Some RNIC designs [5] [10] improve performance by storing context entries on-chip, but these architectures support only a limited number of connections. By scheduling and managing the QPC, csRNA [7] resolves the head-of-line blocking issue in the QPC cache. However, this does not apply to other contexts. ScalaRNIC [6] divides the cache into high-priority and low-priority regions to improve the RNIC's cache hit rate. However, the system performance still degrades for low-priority memory access requests. Ling Zhang et al. [4] design different cache architectures for read-write and read-only modes and support a two-level on-chip cache hierarchy. However, it incurs high resource overhead due to complex control logic, the L1 cache is relatively small, and the L2 cache performs poorly on misses. Moreover, it does not address the head-of-line blocking issue.

This paper presents GCCache that supports all RNIC contexts. It replaces the Cache module in Fig. 1 and achieves non-blocking, low latency, and on-chip storage resources.

III. OVERALL ARCHITECTURE OF GCCACHE DESIGN

A. Architecture Overview

GCCache consists of three subsystems: the Scheduler Subsystem (green in Fig. 2), the Cache Subsystem (yellow), and the Outstanding Request Processing (ORP) Subsystem (blue).

1) **Scheduler Subsystem:** It comprises the three modules: REQ HANDLER, TARGET HANDLER, and RESP HANDLER. This subsystem manages request and response scheduling and process management.

The REQ HANDLER receives requests from the RNIC pipeline, arbitrates and parses them into target, opcode, pipeline ID, and data. The target is used to determine cache hits. The opcode indicates the operation type: *READ* represents reading from the cache and *WRITE* represents writing to the cache. The pipeline ID identifies the pipeline source for response routing. The data contains the payload. The requests are then stored in a request FIFO, awaiting arbitration before being forwarded to the TARGET HANDLER. The RESP HANDLER receives responses from the cache and stores them in the response FIFO of the corresponding pipeline based on the pipeline ID. The TARGET HANDLER initializes a set of Process IDs (PIDs) in a PID Pool. When a new request arrives, a PID is popped from the head of the PID Pool. When the request is processed, the PID is pushed back to the tail. The PID serves as a unique request identifier, enabling pipelined and non-blocking cache access.

2) **Cache Subsystem:** It includes a core state machine, three parallel processing modules (hit processing, replacement processing, status processing), and RAM for data and status storage. This subsystem determines whether the target hits in the cache, computes replacement results based on the replacement algorithm, and maintains cache states.

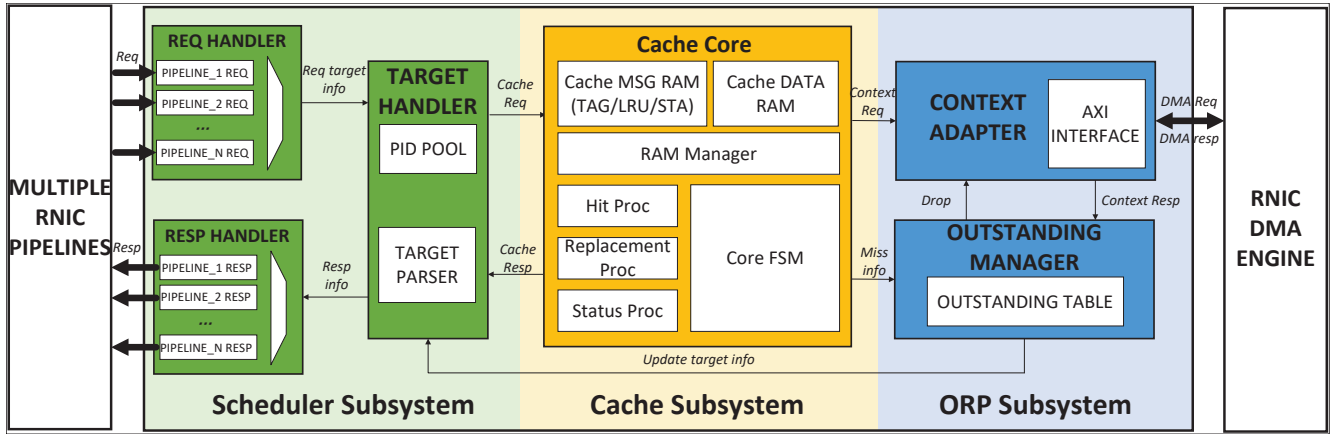


Fig. 2: The GCCache Architecture.

The core state machine reads the data in the cache and the state in the cache when receiving a request. Based on the operation type and cache state, it selectively processes results from the hit, replacement, and state modules, performs corresponding actions, and outputs information or data to downstream modules. If the hit processing module indicates a hit, the status processing module updates the cache status, and hit information is output. If a miss occurs, the cache state is set to WAIT_RESP until the host response returns, and miss information is sent to the ORP subsystem.

3) **ORP Subsystem:** It comprises the two modules: OUTSTANDING MANAGER and CONTEXT ADAPTER. This subsystem records information about missed requests, sends DMA requests to the host, and receives responses.

The CONTEXT ADAPTER interfaces with the DMA engine, receives miss requests from the Cache Core, and initiates DMA requests to fetch contexts from host memory. The OUTSTANDING MANAGER maintains an outstanding table that records miss requests and their order by thread ID. When a response returns via the CONTEXT ADAPTER, the OUTSTANDING MANAGER retrieves the corresponding miss information and loops it back to the Scheduler Subsystem.

GCCache offers strong compatibility across different contexts. Only the code in the CONTEXT ADAPTER module needs to be modified to support a new context, all other modules remain unchanged.

B. Non-blocking and Deduplication Design

In GCCache, hit processing and miss processing are decoupled. The Cache Subsystem handles target resolution. If a target hits, hit information is directly output to the Scheduler Subsystem. If a target misses, miss information triggers the ORP Subsystem. The Cache Subsystem operates with fixed latency. When the ORP Subsystem receives a miss request, it sends a context fetch request to the host. During this process, the Cache Subsystem is not blocked and can accept new requests. If those new requests hit the cache, they can be processed immediately. This decoupling eliminates HOL blocking.

Specifically, miss information from the Cache Subsystem is stored in the outstanding table in the OUTSTANDING MANAGER module by PID. The CONTEXT ADAPTER module sends a read-context request to the host. To avoid blocking, the CONTEXT ADAPTER parallelizes request sending and response waiting, allowing multiple outstanding requests. To associate responses with the correct process, the CONTEXT ADAPTER module needs to store the PID (process ID) in the AXI ID field of the AXI bus. When the response returns, the PID is used to retrieve the corresponding miss information from the outstanding table and forward it to the Scheduler Subsystem.

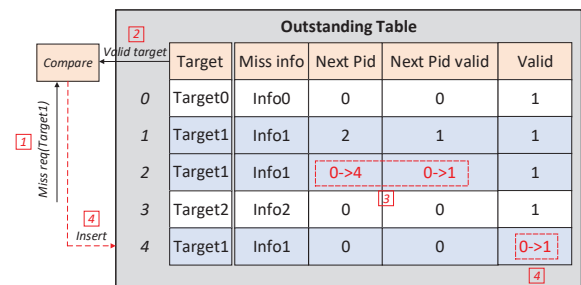


Fig. 3: Processing of an outstanding request table receiving a request, illustrated with a Miss request of target1.

If a duplicate miss request arrives while a context fetch is still outstanding, the OUTSTANDING MANAGER compares the target with all valid entries in the outstanding table. If a match is found, the CONTEXT ADAPTER is inhibited from sending a duplicate request. Instead, the new request is linked to the existing entry via the *next_pid* field. This prevents redundant host accesses. Algorithm 1 describes the processing of Fig. 3. For example, the outstanding table already contains entries for PID0-3. When a new request (PID4, Target1) arrives, it is compared with existing entries. Since PID1 and PID2 have the same target and PID2's *next_pid_valid* is 0, PID2's *next_pid* is set to 4, and *next_pid_valid* is set to 1. PID4's entry is created with *valid*=1 and *next_pid_valid*=0.

Algorithm 1 Outstanding Request Processing

```

1: if input Request is valid then
2:   Insert entry with current pid as index;
3:   Compare valid Target;
4:   Set valid field to 1;
5:   if hit then
6:     Drop context request;
7:     while next_pid_valid == 1 do
8:       if next_pid_valid == 0 then
9:         Set next_pid_valid field to 1;
10:        Set next_pid field to current pid;
11:        Break;
12:       end if
13:     end while
14:   end if
15: end if

```

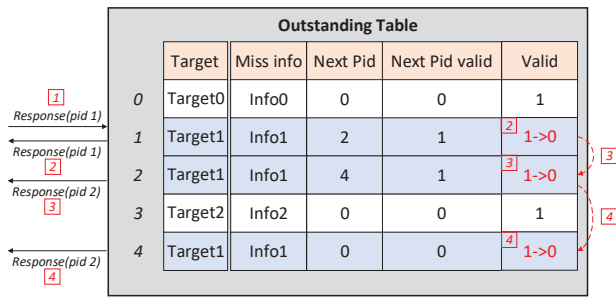


Fig. 4: Processing of outstanding table receiving a response, illustrated with a response of pid1.

Algorithm 2 Response Processing

```

1: if input Response is valid then
2:   Select entry with pid as index;
3:   Send entry to scheduler subsystem;
4:   Set valid field to 0;
5:   while next_pid_valid == 1 do
6:     Select entry with next pid as index;
7:     Send entry to scheduler subsystem;
8:     Set valid field to 0;
9:     if next_pid_valid == 0 then
10:      Break;
11:     end if
12:   end while
13: end if

```

Since only one host request is sent per unique target, only one response is expected. To serve all duplicate miss requests from one response, the OUTSTANDING MANAGER traverses the chain of duplicate requests linked via the *next_pid*. Algorithm 2 describes the response processing. For example, in Fig. 4, when the response for PID1 arrives, its information is output, and *valid* is set to 0. Since *next_pid_valid* is 1, the request for PID2 is served next, and so on, until PID4 (*next_pid_valid* is 0). This ensures all duplicate requests are

served in order from a single host response.

IV. IMPLEMENTATION AND EVALUATION RESULTS

Since commercial NICs are integrated circuits with fixed cache architectures and existing hardware-implemented designs do not disclose implementation details, we have implemented an RNIC with proprietary intellectual property that integrates GCCache. The design was simulated using Cocotb [11] and implemented on the U200 platform, which features a PCIe 3.0 $\times 16$ interface and a QSFP28 100G Ethernet interface.

A. Implementation

TABLE I: GCCache Resource Usage

(a) U200 Resource Usage

	LUTs	Registers	BRAM
QPC	16963(1.43%)	18253(0.77%)	58(2.69%)
CQC	4173(0.35%)	3998(0.17%)	12(0.56%)
SRQC	4185(0.35%)	4064(0.17%)	14(0.65%)
MPT	6550(0.55%)	7962(0.34%)	30(1.39%)
MTT	5704(0.48%)	8331(0.35%)	38(1.76%)
Total	37575(3.16%)	42608(1.80%)	152(7.05%)

(b) Storage Space For On-Chip Communication (MB)

	Research	QPC	CQC	SRQC	MPT	MTT	Total
SRNIC [5]	2.3	-	-	-	-	1.2	3.5
ScalaRNIC [6]	-	-	-	-	-	-	2.16
CX-6	-	-	-	-	-	-	2.0
csRNA [7]	1.4	-	-	-	-	-	-
Paper [4]	0.58	0.038	-	-	-	0.077	0.695
GCCache	0.22	0.033	0.041	0.11	0.13	0.534	

The MTT cache is configured as 4-way 512-set caches with 8 cache lines per set. Other context caches are configured as 4-way 512-set caches with 1 cache line per set. The entire cache subsystem consumes only 3.16% of the LUTs, 1.80% of the registers, and 7.05% of the BRAM resources on the Xilinx U200 FPGA. Compared to existing designs, our architecture achieves significantly lower resource utilization. Detailed resource breakdown is presented in Table I.

B. Evaluation

We integrated GCCache into our proprietary RNIC, supporting caching for QPC, CQC, SRQC, MPT, and MTT. A detailed functional comparison is provided in Table II. In terms of compatibility, except for Mellanox CX-6, all prior works support only a subset of context types, whereas GCCache supports all of them. In terms of non-blocking characteristics, Ling Zhang et al. [4], SRNIC [5] and ScalaRNIC [6] do not support non-blocking access. Although ERNIC [10] avoids head-of-line blocking by storing all context on-chip, this approach severely limits scalability. Moreover, csRNA [7] only addresses HOL blocking for QPC and does not support other context types. In terms of implementation, ScalaRNIC [6] and csRNA [7] were only evaluated in simulation, whereas GCCache has been implemented in hardware.

The cache latency comparison is shown in Fig. 5. Under workloads with 128 and 512 QPs, all of our context caches

TABLE II: Comparison of Cache Functions

Research	QPC	CQC	SRQC	MPT	MTT	Prefetch	Non-block	Hardware implementation
ERNIC [10]	✓	✓	✗	✓	✗	✗	-	✓
ScalaRNIC [6]	✓	✓	✗	✓	✓	✗	✗	✗
csRNA [7]	✓	✗	✗	✗	✗	✗	✓	✗
CX-6	✓	✓	✓	✓	✓	-	-	✓
SRNIC [5]	✓	✓	✗	✗	✓	✗	✗	✓
Paper [4]	✓	✓	✗	✓	✓	✓	✗	✓
GCCache	✓	✓	✓	✓	✓	✓	✓	✓

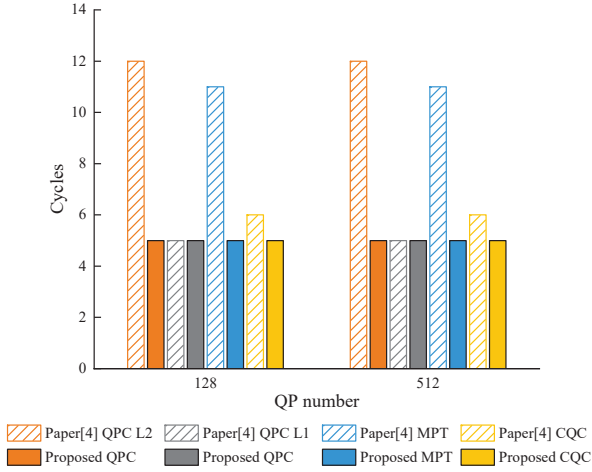


Fig. 5: Cache Request Latency Comparison.

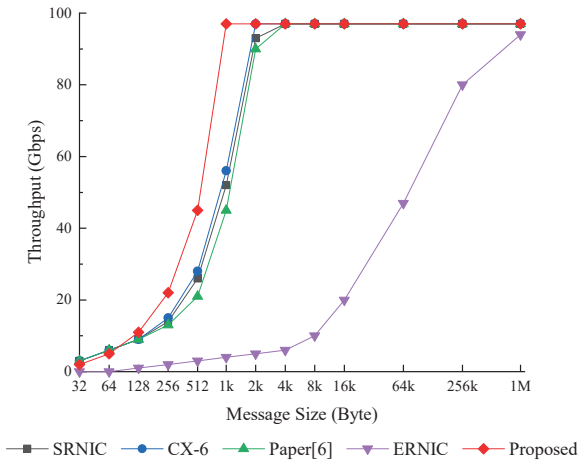


Fig. 6: Throughput of a single QP.

maintain a fixed latency of 5 clock cycles. In contrast, Ling Zhang et al. [4] exhibit latencies exceeding 10 clock cycles for all context types except the first-level QPC cache and the CQC cache.

We evaluated the throughput of our RNIC under a single QP with varying message sizes, as illustrated in Fig. 6. The throughput performance of the RNIC with integrated GCCache is comparable to that of commercial network cards

and significantly outperforms the Xilinx ERNIC. Due to the low latency of GCCache, when the message size exceeds 1 KB, the throughput of the RNIC has approached 100 Gbps.

V. CONCLUSION

This paper addresses the critical challenges of RNIC context caching by proposing and implementing a non-blocking, low-latency general-purpose cache architecture. By decoupling hit processing and miss processing, we eliminate HOL blocking. A deduplication mechanism for outstanding requests reduces PCIe bandwidth waste caused by duplicate misses. Hardware pipelining and parallel processing limit cache response latency to 5 clock cycles, meeting the low-latency requirements of 100 Gbps processing. The architecture supports all types of context caching and is also suitable for 400 Gbps RDMA applications.

ACKNOWLEDGMENT

This work was supported by the Independent Deployment Project of the Institute of Acoustics, Chinese Academy of Sciences: Research on Key Technologies and Equipment Development for Low-Latency Interconnection Networks in Intelligent Computing Center Clusters under Grant MBDX202401. The corresponding author is Zhichuan Guo.

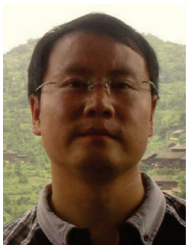
REFERENCES

- [1] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 57–70. [Online]. Available: <https://doi.org/10.1145/3651890.3672233>
- [2] J. Hu, H. Shen, X. Liu, and J. Wang, "Rdma transports in datacenter networks: Survey," *IEEE Network*, vol. 38, no. 6, pp. 380–387, 2024.
- [3] J. Huang, X. Ouyang, J. Jose, M. Wasi-ur Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda, "High-performance design of hbase with rdma over infiniband," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 774–785.
- [4] L. Zhang, X. Yang, Z. Wan, H. Liu, W. Gu, P. Liu, Q. Dai, S. Ye, and Y. Lin, "A high-performance rdma nic with ultrahighly scalable connections," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 6, pp. 2156–2167, 2025.
- [5] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A scalable architecture for RDMA NICs," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1–14. [Online]. Available: <https://www.usenix.org/conference/nsdi23/presentation/wang-zilong>
- [6] X. Ma, F. Yang, Z. Wang, N. Kang, G. Yuan, and X. An, "A scalable rdma network interface card with efficient cache management," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.

- [7] N. Kang, Z. Wang, F. Yang, X. Ma, Z. Ma, G. Yuan, and G. Tan, "csrna: Connection-scalable rdma nic architecture in datacenter environment," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 398–406.
- [8] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, Jun. 2016, pp. 437–450. [Online]. Available: <https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia>
- [9] J. Zhang, H. Yuan, Z. Wang, P. Yu, H. Song, D. Qu, S. Yan, and X. Zheng, "Revisiting the underlying causes of rdma scalability issues," in *2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2024, pp. 227–235.
- [10] Xilinx. (2024) Xilinx embedded rdma enabled nic v4.0 logicore ip product guide. [Online]. Available: <https://docs.xilinx.com/r/en-US/pg332-ernic>
- [11] F. Foundation. (2024) Cocotb. [Online]. Available: <https://www.cocotb.org>



Zhixiang Zhao received the B.S. degree in communication engineering from Northeastern University, Qinhuangdao, China, in 2021, and he is currently pursuing the Ph.D. degree at the school of electronic, electrical and communication engineering of the University of Chinese Academy of Sciences. His current research interests include RDMA-based SmartNIC design and FPGA-based network function offloading.



Zhichuan Guo received the B.S. degree from Wuhan University in 1996, and the Ph.D. degree from the University of Science and Technology of China in 2006. From 1996 to 2003, he served as an Electronics Engineer with the 13th Research Institute of China Electronics Technology Group Corporation and a SDH hardware R&D system engineer of optical networks at Huawei. In 2006, he joined with the Institute of Acoustics, Chinese Academy of Sciences, Beijing, China. Now he is a Professor of CAS engaging in field programmable gate array (FPGA)-based code acceleration, VLSI, RDMA and security.



Mangu Song received the M.Sc. degree in Electronics and Communication Engineering from the School of Microelectronics, Chinese Academy of Science, Beijing, China. Currently she is an assistant research fellow in National Network New Media Engineering Research Center, the Institute of Acoustics of Chinese Academy of Sciences. Her research interests include field programmable gate array (FPGA)-based code acceleration and SmartNIC.



Zhiyuan Ling received his B.S. degree in electronic and information engineering from North China Electric Power University in 2018, and his Ph.D. degree in signal and information processing from Institute of Acoustics, Chinese Academy of Sciences (IACAS) in 2023. Currently he is an assistant research fellow in National Network New Media Engineering Research Center, IACAS. His research interests include RDMA, advanced network and SDN technology.