SPB: Towards Low-Latency CXL Memory via Speculative Protocol Bypassing

Junbum Park†, Yongho Lee‡, Sungbin Jang‡, Wonyoung Lee†, and Seokin Hong‡
†*Dept. of Semiconductor and Display Engineering*, ‡*Dept. of Electrical and Computer Engineering*,
Sungkyunkwan University
†*Memory Division, Samsung Electronics*
Email: {jbrara.park, jhyn205, sunbi3361, wong.lee, seokin}@skku.edu

***Abstract*—Compute Express Link (CXL) is an advanced interconnect standard designed to facilitate high-speed communication between CPUs, accelerators, and memory devices, making it well-suited for data-intensive applications such as machine learning and real-time analytics. Despite its advantages, CXL memory encounters significant latency challenges due to the complex hierarchy of protocol layers, which can adversely impact performance in latency-sensitive scenarios. To address this issue, we introduce the Speculative Protocol Bypassing (SPB) architecture, which aims to minimize latency during read operations by speculatively bypassing several protocol layers of CXL. To achieve this, SPB employs the Snooper mechanism, which extracts essential read commands from the packet data at an early stage, allowing it to bypass multiple protocol layers and reduce memory access time. Additionally, the Hazard Filter (HF) prevents Read-After-Write (RAW) hazards between read and write operations, thereby maintaining data integrity and ensuring system reliability. The SPB architecture effectively optimizes CXL memory access latency, providing a robust solution for high-performance computing environments that require low latency as well as high efficiency. Its minimal hardware overhead makes it a practical and scalable enhancement for future CXL-based memory.**

***Index Terms*—Compute Express Link, CXL memory, Latency, Performance**

I. Introduction

In modern server systems, the growing number of concurrent applications are demanding increasingly larger memory capacity. These applications include machine learning, artificial intelligence, and in-memory database technologies. Compute Express Link [1] (CXL) has emerged as a promising interconnect standard designed to address these requirements by enabling efficient communication between CPUs, accelerators, and memory devices through a flexible, high-speed interface. CXL memory offers significant advancements in memory capacity and sharing capabilities. However, they also introduce new challenges, especially in terms of latency, which can adversely impact overall system performance [2], [3].

Traditional memory systems, such as DRAM-based Dual In-line Memory Modules (DIMMs), have been optimized for direct processor integration with minimal latency overhead. In contrast, CXL-based memory, despite their full-duplex communication capabilities and flexible architecture, incur additional latency [4] due to the complex hierarchy of protocol layers they must traverse. This includes the CXL flex bus layers, which introduce significant delays as they process memory access requests across various protocol layers, resulting in substantial overhead during read and write operations. Consequently, the total access latency in CXL memory is significantly higher compared to traditional DIMMs, posing a major bottleneck for latency-sensitive applications.

To address this issue, we propose the **Speculative Protocol Bypassing (SPB)** architecture, a novel approach that aims to minimize the latency overhead associated with CXL memory. SPB **speculatively bypasses the CXL flex bus layers** for read commands, thereby reducing the overall transit time within the CXL memory stack. By extracting critical command information at an early stage, SPB effectively skips multiple protocol layers, leading to a notable decrease in memory access latency. Additionally, SPB incorporates a Hazard Filter (HF) to prevent read-after-write (RAW) hazards, ensuring data integrity and maintaining system reliability even under speculative execution conditions.

In our study, we conducted real-system measurements using a Intel server [5] equipped with an ASIC-based CXL memory [6] device to accurately capture the latency characteristics of CXL memory. By utilizing parameters obtained from this real-world setup, we could precisely evaluate the performance impact of SPB under practical operating conditions. Our experimental results indicate that SPB achieves up to a 14% increase in execution speed for memory-intensive workloads compared to legacy CXL memory. Furthermore, SPB shows an average of 14% reduction in average memory access time (AMAT) without incurring significant hardware overhead, making it an effective solution for optimizing CXL memory performance.

In this paper, we present a detailed analysis of the latency breakdown in CXL memory and introduce the SPB architecture as a means to minimize this latency. We explore the design and implementation of SPB, including its snooper and hazard filter mechanisms, and evaluate its effectiveness through comprehensive experiments using Ramulator2 [7]. Our results highlight the potential of SPB to significantly enhance the performance of CXL memory, enabling the way for more efficient and responsive computing environments in the future.

II. Background

A. CXL Protocol

The CXL protocol, as shown in Figure 1, is structured into multiple layers and inherits the functional elements of traditional PCIe [8]. It introduces new features such as CXL.cache

978-3-9826741-0-0/DATE25/© 2025 EDAA

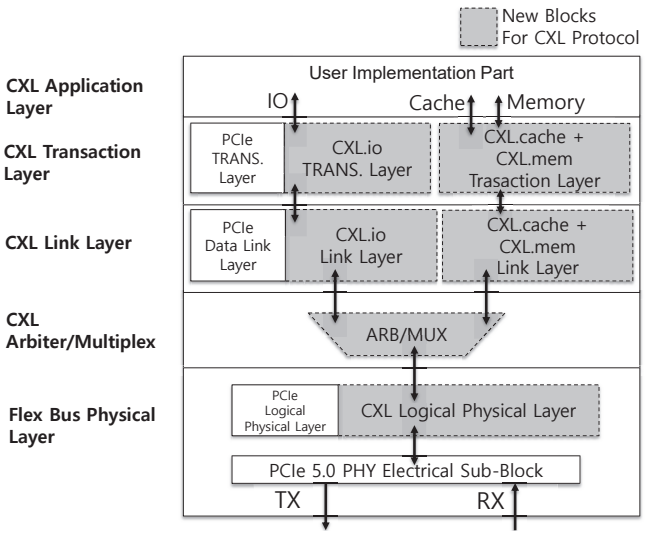

Fig. 1. CXL Flex Bus Layers

and CXL.mem to enhance its capabilities. CXL.mem provides byte-level memory access, while CXL.cache allows the device's address space to be used as a cacheable region. Together, these enhancements enable CXL protocol to address the need for a unified high-performance interconnect that adapts to the diverse requirements of modern computing environments.

Designed to support accelerators and memory devices through a high-bandwidth, low-latency channel, the CXL protocol includes CXL.io, which is responsible for device discovery, enumeration, and error reporting, along with the optionally implemented CXL.cache and CXL.mem protocols for memory access.

B. The Flit

Flow control units (Flits) are packets created specifically to support the CXL protocol and are used for transmitting data within it, as shown in Figure 2. This packet, with a fixed size of 68 bytes, plays a key role in reducing the latency of the CXL protocol. In contrast, the payload size of PCIe's Transaction Layer Packet (TLP) ranges from 128 bytes to 4 KB, which incurs significant latency in determining and interpreting the TLP size. CXL addresses this issue by utilizing packets with a fixed size, thereby eliminating the additional latency. Each Flit consists of a 2-byte Protocol ID, a 64-byte payload, and a 2-byte CRC. There are four types of Flits, each identified by a unique Protocol ID: CXL.io Flits, which transport TLPs and DLLPs (Data Link Layer Packets) within the Flit payload; CXL.cache/CXL.mem Flits, which include protocol Flits and control Flits; ALMP Flits, representing ARM/MUX Layer Link Management Packets; and NULL Flits. Protocol Flits are specifically used for read and write commands. Each Flit contains a Flit type field to indicate whether it is a Control Flit or a Protocol Flit, a Size (Sz) field to determine whether 64B or 32B of data will be transmitted, and Slot 0, 1, and 2 fields to specify the type of each slot. Among the slot types, read and write commands, as well as write and read data, are included. A Protocol Flit consists of four slots, each capable of containing a command. Furthermore, to reduce the

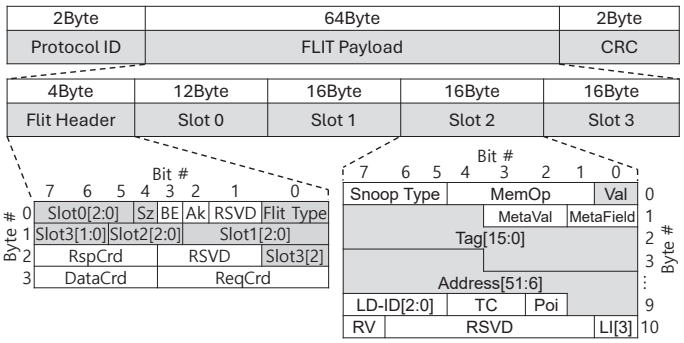

Fig. 2. Flit layout

implementation complexity of the CXL flex bus layer's Physical Layer, the specification [9] limits the number of commands per Flit to a maximum of two reads and one write.

C. System Architecture with CXL memory

CXL memory (CXM), also known as a CXL type-3 device, utilizes only CXL.mem and CXL.io, excluding CXL.cache. This device type primarily operates using CXL.mem to service requests sent from the host. As shown in Figure 3, the CXL device introduces a CXL Root/Device port interface and a connecting link for the CXL interface, unlike the traditional DRAM devices directly connected to the processor core(s). The CXL Flex Bus Layers, as shown in Figure 1, symmetrically exists within both the Root and Device Port interfaces. As each layer is traversed, the latency of a CXL memory device using the CXL interface inevitably increases. Additionally, as the length of the link connecting the Port and Device increases, the data transmission time correspondingly increases.

III. Motivation: CXL Memory Latency

Our motivation comes from an important observation regarding the performance result associated with the latency of CXM. In particular, through real system measurements using the latest Intel server [5] equipped with one of the most recent ASIC-based CXM [6], with detailed specifications provided in Table I.

We identified significant overhead in processor performance, particularly in load instructions, due to the high latency of the CXM.

Overall CXL Memory Latency: To measure the latency of both DRAM and the CXM, we used the Memory Latency Checker (MLC) [10]. As shown in Table II, NUMA node 0 was allocated to DRAM, while NUMA node 1 was assigned to the CXM. Our measurements indicated that CPU access to memory on node 0 takes 149.5 ns, whereas accessing memory on node 1 takes 285.2 ns. This 285.2 ns includes overhead from the PCIe bus connection and the CXL device. To isolate the device's pure latency, we connected a protocol analyzer[1]

[1]The CXL protocol analyzer is monitoring in/out packet of CXL device with interposer equipment [11].

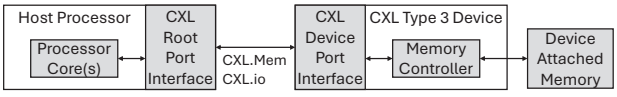

Fig. 3. System architecture with CXL memory

TABLE I
SYSTEM CONFIGURATIONS

Server system	
Component	**Description**
OS (kernel)	Ubuntu 22.04.2 LTS (Linux kernel v6.8.2)
CPU	Intel®Xeon Family 6 CPUs @2.4 GHz, 56 cores and 288MB LLC, Hyper-Threading Enabled
Memory	DDR5-4800 (64GB)
CXL memory	
Controller type	ASIC
Interface speed	PCIe Express Gen.5 x8
Protocol	CXL 2.0
Memory	DDR5-4800 (256GB)

TABLE II
LATENCY FOR SEQUENTIAL MEMORY ACCESSES

NUMA Node	0 (DRAM)	1 (CXL memory)
PCIe Wire Delay (ns)	N/A	135.2
Device (ns)	149.5	150
Total Latency (ns)	149.5	285.2

directly in front of the device, revealing an input-to-output pin latency of 150 ns, with the remaining 135.2 ns attributed to the PCIe connection wire delay.

Breakdown of CXL Memory Latency: To identify areas where latency reduction is possible, we analyzed the internal components of the CXL device, including the CXL flex bus layer, internal bus, DRAM controller, and DRAM itself. Based on our estimations, the latency contributions are as follows: 70ns for the CXL flex bus layer (CXL Device Port Interface), 20ns for the internal bus, and 60ns for both the DRAM controller and DRAM. This value is an estimated prediction based on the range of values mentioned in various papers [12], [13], [14]. In the case of internal bus latency, the delay is caused by the physical distance between IP blocks within the device, which arises during chip implementation due to place and route. Optimizing this architecturally is beyond the scope of this paper. Regarding the DRAM controller and DRAM, optimal configurations for DDR5 support have already been implemented. With these conditions in mind, we focused on exploring optimization strategies for the CXL flex bus layer, as it contributes the most to overall latency.

Motivational Insight: Our attention then shifted to the characteristics of the read command. Since a read command does not modify data, even if it is performed at an incorrect location, no issues arise as long as the system does not receive incorrect data. Based on this, we investigated where the read command is first identified, discovering that this occurs at the interface where the Flit transitions from the Physical layer to the ARB/MUX layer. By sending only the essential information from the Flit to the DRAM controller at this point shown in Figure 4, it became clear that latency could be significantly reduced. *This insight led to the development of a novel architecture.*

IV. SPECULATIVE PROTOCOL BYPASSING

In this section, we introduce ***Speculative Protocol Bypassing (SPB) architecture*** that *bypasses the CXL flex bus layers* for the read commands. SPB effectively reduces the Flit transit time in the CXL flex bus layers while maintaining their full functionality. To achieve this, SPB employs a ***snooper*** that monitors the Flits on the physical CXL link and then speculatively skips the Flit processing time across the link layer, the transaction layer and the application layer, as illustrated in Figure 4. While this approach reduces the flip processing time, it can introduce a read-after-write (RAW) hazard. To mitigate this, SPB employs a ***hazard filter (HF)*** that tracks these addresses.

A. Speculative Protocol Bypassing with Snooper

The CXL flex bus layers process the Flits by passing them through multiple layers to ensure proper and secure data transmission. This processing introduces additional latency to the CXL memory accesses (the specific latency of each stage is detailed in Figure 5). The first position where the Flit sent by the host can be identified within the CXL memory (CXM) is just after decoding the information received from the physical layer. This position corresponds to the interface between the ARB/MUX and the physical layer, as shown in Figure4. At this position, the Flit has not yet undergone integrity verification, which is performed later at the link layer.

Despite the lack of integrity verification for the Flits, the read/write commands for the DRAM controller can be extracted from the Flits just after the physical layer and used to access the DRAM devices, potentially reducing the latency by 20%. To leverage this speculative approach, SPB employs a snooper that extracts the read/write information from the Flits received from the physical layer. The Flit processing time from the ARB/MUX to the application layer is approximately 30ns, as shown in Figure 5. In contrast, the latency of a snoop path is only 3ns as the snooper quickly extracts the read/write information. This rapid information extraction is possible because the snooper extracts the information by comparing the Flit data with predefined patterns, as will be detailed in Section IV-C.

B. Preventing RAW Hazards with Hazard Filter

A potential risk of using the snooper to initiate a read command earlier is the occurrence of read-after-write (RAW) hazards. These hazards occur when a read command accesses a memory address that is still being written to. For example, if a write operation is performed on address 0x100, followed immediately by a read operation, the snoop path passes the read command to the DRAM controller before the corresponding write command has reached the DRAM controller. This happens because the write command is only forwarded to the DRAM controller after the write data has arrived at the hazard filter via the normal path. As a result, the read operation would return outdated (i.e., old) data rather than the newly updated

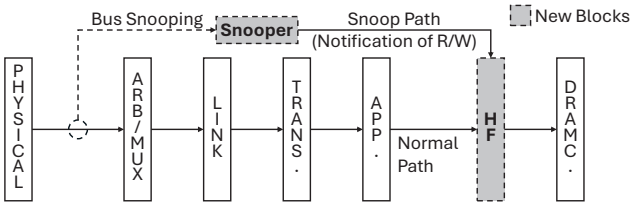

Fig. 4. Overview of speculative protocol bypassing (SPB)

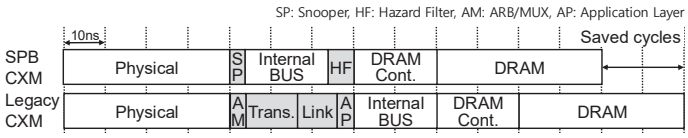

Fig. 5. Latency reduction with speculative protocol bypassing

data for the given memory address, potentially compromising data integrity.

SPB employs a hazard filter to address the RAW hazard issue. The hazard filter stores information about the outstanding write commands and their corresponding address ranges within the device. When it receives a new read command, it compares the address range with the previously stored ones. If there is an overlapping region, the read request arriving through the snoop path is not forwarded to the DRAM controller. When the read command later arrives at the hazard filter through the normal path, the overlapping write command has already passed through the hazard filter and reached the DRAM controller. This allows the read command to be forwarded to the DRAM controller without causing a RAW hazard. The execution time of the hazard filter, including command registration, deletion, and lookup functions, is 7ns. Therefore, the total latency reduction achievable using the SPB is 20ns.

C. Implementation of Snooper

The snooper extracts only the essential information from the Flit required to process read and write commands and quickly forwards it to the hazard filter. Since a Flit contains a significant amount of information, proper decoding and matching of its format are required to extract the necessary information.

Figure 6 illustrates the flow of the extraction algorithm. Upon the arrival of a new Flit via the bus snooping path, the snooper checks three fields: *Protocol ID*, *Flit type*, and *Slot format*. The snooper first examines the Protocol ID to identify CXL.mem Flits. Once identified, it checks the Flit type field in the Flit header. If the Flit type is a protocol Flit, the snooper then inspects each Slot format field (i.e., slot0 - 3) in the Flit header to identify the slots containing read or write commands. A "CXL.Mem M2S RwD" slot format indicates that the corresponding slot contains a write command, while a "CXL.Mem M2S Req" format signifies that the corresponding slot contains a read command. The size of the transmitted data is determined by the Sz field in the Flit header (1 representing 64B, 0 representing 32B).

If a Flit containing read and/or write commands is successfully identified, the snooper generates a corresponding notification; otherwise, the Flit is discarded. Although the extraction process is described sequentially above, in hardware implementation, all necessary fields of the Flit are examined simultaneously and matched with a specific pattern. This simultaneous matching enables rapid extraction of read/write commands.

D. Implementation of Hazard Filter

To prevent Read After Write (RAW) hazards, the hazard filter utilizes two tables: the Write Region Table (WRT) and the Read Mark Table (RMT). These tables work together to ensure proper sequencing and data integrity.

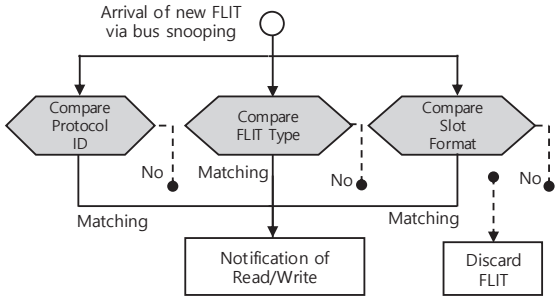

Fig. 6. An algorithm for extracting read/write command information

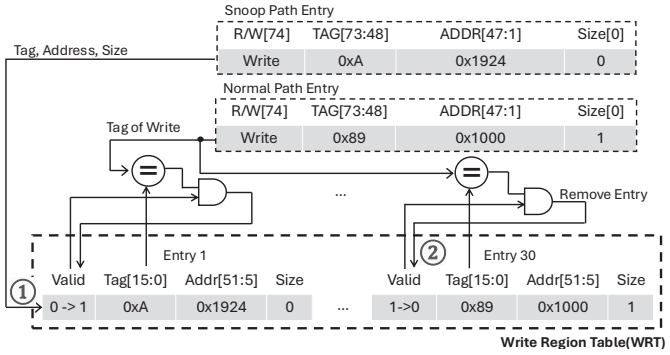

Fig. 7. Write command flow of Hazard Filter

The WRT, illustrated in Figure 7, monitors potential overlaps between read and write commands. A WRT entry is created when a write command arrives through the snoop path, setting the valid bit to indicate an active entry (①). Once the same command is received via the normal path, the entry is invalidated, and the valid bit is cleared (②).

Case 1: A read command is not overlapped with a write region. When a read command arrives via the snoop path, it compares its address and size with the entries in the WRT (①) as illustrated in Fig. 8. If no overlap is detected, a speculative read request is immediately sent to the DRAM controller (②). The corresponding read command's information, including its tag, is then recorded in the Read Mark Table (RMT) with the mark bit set to 0 (③). This ensures that the speculative read has been performed. The overlap detection mechanism is depicted in Figure 8, where the address and size of the incoming read command are checked against the valid entries in the WRT to confirm no conflict with outstanding write command regions.

When the read command arrives via the normal path, the hazard filter checks the RMT for a matching entry based on the tag (④). If an entry is found and its mark field is still set to 0, it indicates that a speculative read has already been issued, and no additional action is necessary (⑤). This process prevents redundant read operations and ensures data integrity.

Case 2: A read command is overlapped with a write region. When a read command arrives via the snoop path, it compares its address region with the valid entries in the WRT. If an overlap is detected, indicating a potential RAW hazard, the read command is not sent to the DRAM controller to avoid

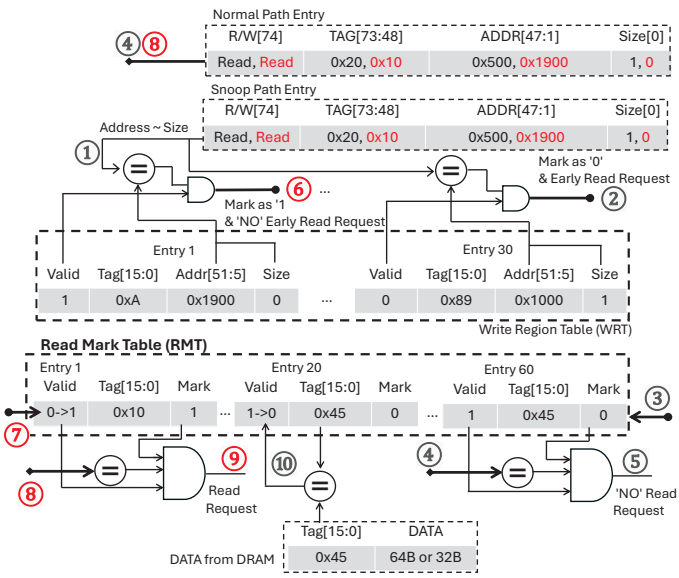

Fig. 8. Read command flow of Hazard Filter

such hazards (⑥). This decision is recorded using a field called the mark bit, which is set to 1 in the corresponding RMT entry along with the read command's tag (⑦).

Subsequently, when the same read command arrives through the normal path (⑧), all write commands that could potentially cause a hazard have already been processed by the DRAM controller. As a result, the read can be safely executed without causing a RAW hazard (⑨). This process is carried out by checking the RMT for an entry with the same tag and confirming that the mark bit is set to 0. This mechanism is illustrated in Fig. 8.

Case 3: When the read-requested data arrives from the DRAM controller. The matching entry in the RMT is found by its tag, the entry is deleted, and the data is passed to the application layer to complete the execution of the read command (⑩) as illustrated in Fig. 8.

E. Handling Mis-speculation

Mis-speculation can occur if the Flit data becomes corrupted during transmission through the physical CXL link. Detecting corrupted Flit typically points to a faulty connection with the host system, which undermines performance optimization. Although the likelihood of receiving an erroneous Flit is low, ensuring the functional correctness of the SPB is essential.

A Flit arriving through the snoop path can cause several issues, including: 1) misidentification of read/write commands, 2) incorrect memory addresses, and 3) corrupted tags. These problems may result in erroneous command registration in the hazard filter, compromising system accuracy.

To mitigate this, the HF employs a history queue to track the tags of commands received through the snoop path. If an error occurs in a flit, it is detected by the Link layer using a CRC check. The erroneous flit is discarded, and this information is communicated to the host via the CXL.io protocol path. Consequently, the command that arrived at the HF through the snoop path will not reach the normal path.

When a command arrives via the snoop path, its tag is stored in the snoop path history queue, which keeps track of the history of commands received through this path. If the same command later arrives through the normal path, the history queue is consulted to retrieve the corresponding entry and compare it with the snoop path command. If they do not match, all actions associated with the tag from the history queue entry are discarded.

To achieve this, the HF searches for the same tag in both the Write Response Table (WRT) and the Read Response Table (RMT). If the tag is found in the WRT, the corresponding entry is deleted. If the tag is found in the RMT, the entry is deleted as well, and any incoming read data associated with that tag is discarded. If multiple errors occur consecutively, the recovery process repeats until the entries in the normal and snoop paths match, ensuring system stability.

V. EVALUATION

A. Experimental Methodology

We carry out architectural experiments using the fast, cycle-accurate DRAM simulator Ramulator2 [7]. To emulate the CXL memory controller, a CXL flex bus layer was placed between the DRAM Controller and the LLC, where queues were created to store read/write commands, and additional latency was introduced at this layer. Detailed architecture parameters are given in Table III; In this paper, the baseline architecture is Legacy CXM. We use ten integer benchmarks from SPEC CPU2017 [15] suit. For all benchmarks, we fast-forward 10 billion instructions and perform cycle-accurate simulation for next 1 billion instructions.

The performance of the SPB CXM is evaluated taking into account the probability of handling mis-speculation, based on the Bit Error Rate (BER) specified in the PCIe standards [16], which is 10^{-6}. This rate implies a probability of one bit error per 1,837 Flits. As a result, in the worst-case scenario, an additional read may be necessary every 1,837 reads.

B. Performance

In Figure 9, we compare the execution speed of SPB CXM, Legacy CXM, and their configurations with DRAM, using a 5:5 memory allocation ratio. The left graph highlights a significant average performance improvement of 14% for SPB CXM, showcasing the effectiveness of the proposed approach. Notably, this improvement includes the probability of mis-speculation, with mis-speculation-induced performance degradation being a negligible 0.008%, indicating minimal impact on overall performance. In the 429.mcf benchmark, which has high

TABLE III
SIMULATED SYSTEM CONFIGURATION.

Component	Parameter
CPU	4-core, 4-way Out-of-Order, 3.2GHz
LLC	8MB, 16-way, LRU, 47-cycle
Round-way Latency	150ns (Legacy CXM) 130ns (SPB CXM) 60ns (DRAM)
CXL flex bus layer	1.0Ghz Emulated CXL Controller
DRAM	DDR5-3200AN, 32GB, 2-channel, 2-rank

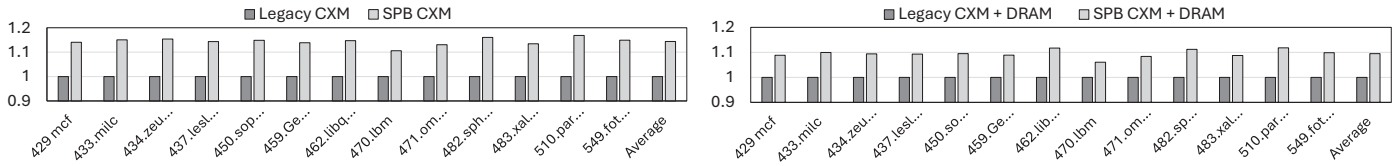

Fig. 9. Execution speed comparison for SPEC Benchmarks. Legacy CXM vs SPB CXM (Left), Legacy CXM + DRAM vs SPB CXM + DRAM (Right), with values normalized based on Legacy CXM

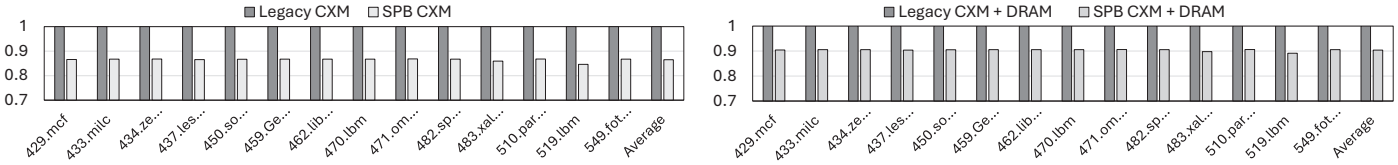

Fig. 10. Average memory access time for SPEC Benchmarks. Legacy CXM vs SPB CXM (Left), Legacy CXM + DRAM vs SPB CXM + DRAM (Right), with values normalized based on Legacy CXM

memory accesses per kilo instructions (MPKI), a 15% performance increase is observed, while 470.lbm, with lower MPKI, shows an 11% improvement. The right graph, illustrating tests with DRAM, indicates a 10% average performance gain for SPB CXM, with high MPKI areas seeing an 11% increase and low MPKI areas around 9%.

In Figure 10, we evaluate the average memory access time (AMAT) for SPB CXM, Legacy CXM, and their configurations with DRAM. The left graph shows a 14% reduction in AMAT for SPB CXM, emphasizing its substantial impact on memory access efficiency. This improvement is attributed to a high LLC read miss rate (>90%) in most benchmarks, which led to the majority of CPU load requests being directed to memory, resulting in performance gains. The right graph shows a 10% AMAT reduction with SPB CXM when used with DRAM, consistent with the observed execution speed improvements.

C. Implementation Costs and Limitations

The Snooper uses combinational logic to compare predefined patterns with the Flit layout. When the Protocol ID, Type, Slot 0, 1, and Valid bit match the specified pattern, it sends the Tag and Address, requiring approximately 1,000 gate counts.

The Hazard Filter (HF) consists of the Write Response Table (WRT), Read Mark Table (RMT), and History Queue. The WRT manages up to 30 entries to account for the 30ns delay between the snoop and normal paths. Each entry includes an address, tag, size, and valid bit, with a total size of 8 bytes per entry, resulting in 240 bytes (8 bytes × 30 entries) and approximately 20,000 gate counts using D flip-flops. Typically, each bit in Samsung's 8nm process technology needs around 10 gate counts for flip-flops.

The RMT stores up to 60 outstanding read commands due to the 60ns DRAM response time, with each entry occupying 64 bytes, resulting in a total of 3.75KB (64 bytes × 60 entries) and requiring approximately 40,000 gate counts. SRAM is used here, as it is more area-efficient for larger storage, requiring roughly 10,000 gate counts per 1KB. Additional flip-flops for storing 60 tags, marks, and valid bits add another 11,000 gate counts.

The History Queue, storing 30 tag entries, also uses flip-flops, adding around 5,000 gate counts for storage and associated control logic.

Overall, the HF implementation requires about 81,000 gate counts. Combined with the Snooper, they account for only around 0.2% of the total chip area, given a typical CXL controller with approximately 40 million gate counts.

VI. CONCLUSION

In this paper, we proposed the Speculative Protocol Bypassing (SPB) to address the latency challenges in CXL memory. SPB reduces memory access latency by bypassing the CXL flex bus layers for read commands and integrating a Hazard Filter (HF) to prevent read-after-write (RAW) hazards, significantly improving system performance in latency-sensitive environments. Based on parameters obtained from real-system measurements, our evaluation shows that SPB achieves up to a 14% increase in execution speed and an average 14% reduction in memory access time compared to legacy CXL memory. These performance improvements are particularly notable for memory-intensive workloads and come with only a minimal hardware overhead, adding 0.2% to the overall CXL controller area.

As CXL technology evolves, the complexity of protocol handling in the flex bus layers will increase, leading to further latency overhead. Consequently, the need for efficient latency optimization techniques like **SPB will become even more critical with each new version of the CXL specification**. The SPB addresses these growing demands by providing a crucial solution for maintaining low latency and high efficiency in high-performance computing environments.

ACKNOWLEDGMENT

This work was partly supported by the Ministry of Science and ICT (MSIT) under the Information Technology Research Center (ITRC) support program (No.II212052, 50%) supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP) and Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.00228970, 50%). Seokin Hong is the corresponding author.

REFERENCES

[1] *Compute Express Link (CXL): All you need to know*, Rambus, 2023. [Online]. Available: https://www.rambus.com/blogs/compute-express-link

[2] D. D. Sharma, R. Blankenship, and D. Berger, "An introduction to the compute express link (cxl) interconnect," *ACM Computing Surveys (ACM Comput. Surv.)*, vol. 56, no. 11, p. Article 290, July 2024.

[3] J. Park, W. Lee, T. Kim, Y. Lee, and S. Hong, "Performance characterization of cxl memory expander: Impact on read and write latencies," in *2024 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2024, pp. 1–5.

[4] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong *et al.*, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, p. 3.

[5] *Products formerly Sapphire Rapids*, Intel, 2023. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/codename/126212/products-formerly-sapphire-rapids.html

[6] *Samsung Develops Industry's First CXL DRAM Supporting CXL 2.0*, Samsung, 2023. [Online]. Available: https://semiconductor.samsung.com/news-events/news/samsung-develops-industrys-first-cxl-dram-supporting-cxl-2-0/

[7] H. Luo, Y. C. Tu, F. N. Bostancı, A. Olgun, A. G. Ya, O. Mutlu *et al.*, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, 2023.

[8] PCI-SIG, "Pci express base specification revision 5.0, version 1.0," *Placeholder Journal*, 2019.

[9] *CXL 3.1 Specification*, Compute Express Link Consortium, 2023. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[10] V. Vish, K. Karthik, W. Thomas, S. Sri, and S. Sharanyan, "Intel memory latency checker v3.11," *Intel*, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html

[11] *CXL Protocol Analysis and Exerciser Tools*, Teledyne LeCroy, 2023. [Online]. Available: https://www.youtube.com/watch?v=I2zlRlcofQw

[12] D. D. Sharma, "Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing," in *IEEE Symposium on High-Performance Interconnects (HOTI)*, 2022, p. 10.

[13] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal *et al.*, "Pond: Cxl-based memory pooling systems for cloud platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, p. 575.

[14] A. Cho, A. Saxena, M. Qureshi, and A. Daglis, "A case for cxl-centric server processors," *arXiv preprint arXiv:2305.05033*, 2023.

[15] *Standard Performance Evaluation Corporation*, SPEC. [Online]. Available: https://www.spec.org/cpu2017

[16] *The PCIe® 6.0 Specification Webinar: Error Detection and Correction with FEC*, pcisig, 2023. [Online]. Available: https://pcisig.com/pcie%C2%AE-60-specification-webinar-qa-error-detection-and-correction-fec