Buddy ECC: Making Cache Mostly Clean in CXL-based Memory Systems for Enhanced Error Correction at Low Cost

Yongho Lee*, Junbum Park†, Osang Kwon*, Sungbin Jang*, and Seokin Hong*
*Department of Electrical and Computer Engineering, †Department of Semiconductor and Display Engineering
Sungkyunkwan University, Suwon, Republic of Korea
{jhyn205, jbrara.park, osang915, sunbi3361, seokin}@skku.edu

***Abstract*—As Compute Express Link (CXL) emerges as a key memory interconnect, interest in optimization opportunities and challenges has grown. However, due to the different characteristics of the CXL Memory Module (CMM) compared to traditional DRAM-based Dual In-line Memory Modules (DIMMs), existing optimizations may not be effectively applied. In this paper, we propose an Buddy ECC that leverages the full-duplex nature and features of the CMM to optimize bandwidth, enhance reliability, and reduce area overhead. First, the Proactively Write-back improves bandwidth efficiency by minimizing dirty cachelines in the last-level cache through dead block prediction, proactively identifying and writing back cachelines that are unlikely to be rewritten. Second, the Utilization-aware Policy dynamically monitors the internal bandwidth of the CMM, sending write-back requests only when the module is under low-load rate, thus preventing performance degradation during high traffic. Finally, the Buddy ECC scheme enhances data reliability by separating Error Detection Code (EDC) for clean cachelines and stronger Error Correction Code (ECC) for dirty cachelines. Buddy ECC improved bandwidth utilization by 46%, limited performance degradation to 0.33%, and kept energy consumption increase under 1%.**

***Index Terms*—Compute Express Link, CMM, Reliability, ECC**

I. Introduction

As the demand for high-performance computing continues to grow, the memory interconnect landscape is undergoing significant transformation. Compute Express Link (CXL) has emerged as a cutting-edge technology [1]–[5], enabling efficient memory sharing and high-bandwidth communication between CPUs, accelerators, and memory modules. This advancement is particularly critical for data-intensive applications, artificial intelligence, and large-scale distributed systems. However, the adoption of CXL-based memory systems, such as the CXL Memory Module (CMM), presents several challenges, particularly in maintaining bandwidth efficiency [4], [6], [7].

Conventional memory systems, including DRAM-based Dual In-Line Memory Modules (DIMM), have been optimized for half-duplex communication, where read and write operations share the same data bus [8], [9]. However, with the introduction of CMM that supports full-duplex communication, new challenges related to bandwidth efficiency have arisen, and optimization strategies are required to fully leverage the capabilities of full-duplex communication in CMM-based systems.

Error correction mechanisms have evolved alongside modern memory systems to ensure data reliability. Conventional cache structures have relied on Single Error Correction-Double Error Detection (SEC-DED) [10]. While effective for early systems, SEC-DED is increasingly insufficient as memory capacities and error rates rise. To provide stronger protection, more robust mechanisms such as Double Error Correction-Triple Error Detection (DEC-TED) have been adopted, offering enhanced error correction capabilities to meet the demands of modern high-speed and large-capacity memory systems [11].

In this paper, we propose ***Buddy ECC***, a novel error correction system specifically designed to address the challenges in CXL-based memory systems, including bandwidth underutilization caused by imbalanced read-write ratios, the increased likelihood of errors in high-speed, high-capacity memory environments, and the need for efficient error correction mechanisms that minimize latency and area overhead while maintaining system reliability and performance. Buddy ECC is designed with two major principles. The first focuses on optimizing bandwidth utilization through *Proactively Write-back* and a *Utilization-aware Policy* to minimize the number of dirty cachelines. The primary concept of the Proactively Write-back mechanism is to optimize CMM bandwidth utilization by proactively write-back cachelines to the CMM that are unlikely to be rewritten. The Utilization-aware Policy dynamically monitors the load on the CMM, triggering write-back operations only during low-load periods to prevent CMM system overload, thus improving both memory efficiency and system performance. The second principle involves separating Error Detection Code (EDC) and Error Correction Code (ECC), where simple parity checks are applied to clean cachelines, while robust ECC mechanisms are reserved for dirty cachelines. This robust ECC is achieved by borrowing the ECC area of clean cachelines to provide additional protection for other dirty cachelines. This approach reduces unnecessary error correction overhead for clean cachelines while providing robust protection for critical data stored in dirty cachelines.

We demonstrate how Buddy ECC addresses the limitations of conventional policy in CXL-based memory systems by enhancing both bandwidth efficiency and data reliability. Through extensive simulations using the ramulator2 [12] simulator, we show that Buddy ECC significantly improves overall memory

978-3-9826741-0-0/DATE25/© 2025 EDAA

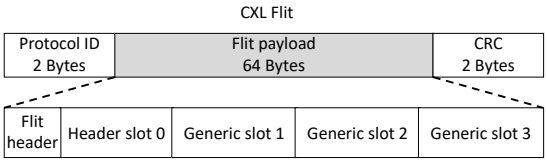


Fig. 1. Structure of a CXL flit with detailed breakdown of the payload and slots.

utilization, reduces the number of dirty cachelines and minimizes performance degradation due to error correction.

The Buddy ECC system, evaluated across 28 SPEC CPU2006 [13] and SPEC CPU2017 [14] workloads, achieves 46% average improvement in bandwidth utilization the baseline, with performance degradation limited to an 0.33% and energy consumption increase kept under 1%.

II. Background

A. Compute Express Link

Compute Express Link (CXL) [1], [15] is emerging as a prominent technology for future memory systems, with prototypes already available on the current market. CXL is an open standard interconnect protocol designed to facilitate high-speed communication between CPUs, memory, and accelerators. It operates over the Peripheral Component Interconnect Express (PCIe) physical layer, leveraging the widespread adoption and compatibility of PCIe infrastructure. CXL supports three key protocols: CXL.io, CXL.cache, and CXL.mem.

The CXL Type-3 Memory Module (CMM) utilizes both the CXL.io and CXL.mem protocols, leveraging flits within PCIe packets for its operations. Figure 1 illustrates the structure of a flit. Flit is composed of three main components: the Protocol ID, Flit Payload, and Cyclic Redundancy Check (CRC). The Protocol ID is a component of the flit structure in CXL, responsible for identifying the specific protocol being used for a given transaction. The CRC is an error detection mechanism embedded within the flit. The flit payload is the main body of the flit, containing the actual data being transmitted between devices. The flit payload is composed of one header slot and three generic slots. Each slot can contain a message packet for communication between the host and CMM. That message packet is specific types of requests designated for slots.

In the context of the CMM, the flit payload can carry multiple types of messages, including read requests (i.e., Master to Subordinate Request w/o Data, hereafter referred to as **M2S Read**) and write request (i.e., Master to Subordinate Request w/ Data, hereafter referred to as **M2S Write**). Specifically, a flit payload can accommodate up to two M2S Read requests and one M2S Write request, a limitation designed to maintain an average read-to-write (R/W) ratio of 2:1 and to reduce the complexity of the receiver [15].

B. Peripheral Component Interconnect Express

Peripheral Component Interconnect Express (PCIe) [16] provides high bandwidth largely due to its full-duplex communication capability, which allows simultaneous data transmission and reception on the same connection. The full-duplex nature allows PCIe to handle data-intensive applications efficiently by supporting high data rates in both directions simultaneously, making it ideal for use in high-performance computing, data centers, and other environments where large volumes of data must be moved quickly [1].

In the context of the CMM, the architecture operates in a full-duplex mode, meaning that data can flow simultaneously in both directions: Master-to-Subordinate (M2S) and Subordinate-to-Master (S2M) [17]. This full-duplex capability is crucial to optimizing the bandwidth utilization and overall performance of the system. However, considering most workloads, the read bandwidth is used more than the write bandwidth. Therefore, since M2S and S2M currently have the same bandwidth, the S2M bandwidth is the bottleneck and the M2S bandwidth is under-utilization [6].

C. Static Random Access Memory

The Static Random Access Memory (SRAM) cache structure is designed to provide fast and reliable data access, consisting of cachelines, tag arrays, and data arrays. Each cacheline, typically 64-byte in size, holds multiple words and is associated with a tag to identify its corresponding memory address [18]. To ensure data integrity, Error Correction Code (ECC) is employed, adding extra bits to each cacheline to detect and correct errors. While Single Error Correction-Double Error Detection (SEC-DED) has been commonly used, there is increasing emphasis on data reliability, leading to the adoption of more advanced schemes such as Double Error Correction-Triple Error Detection (DEC-TED). In Intel's Emerald Rapids architecture [11], the 64-byte cacheline with DEC-TED support incurs a 68-bit overhead, which amounts to approximately 13% [19].

III. Motivation

A. Bandwidth Utilization Between DIMM and CMM

Figure 2 illustrates the comparison of bandwidth efficiency between DRAM-based Dual In-Line Memory Module (DIMM) and CMM in various read-to-write (R/W) ratios, using the Table II environment with Intel Memory Latency Checker (MLC) [20] to compare bandwidth utilization. In the experiments, as the names suggest, all test run on the DIMM for the DIMM tests and on the CMM for the CMM tests. For DIMM, bandwidth efficiency remains high under read-only conditions but gradually decreases as the R/W ratio approaches 1:1. In contrast, the CMM shows lower efficiency for read-only but steadily improves as the R/W ratio becomes more balanced. This behavior can be attributed to DIMM's half-duplex nature, where both read and write operations share the same data bus [8], [9], leading to contention. In contrast, the CMM, which

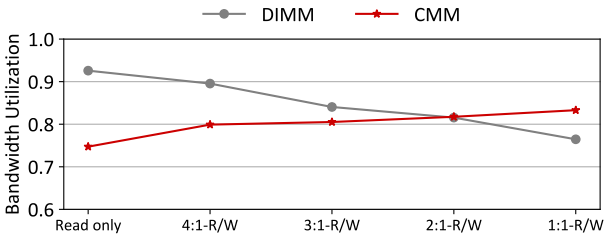
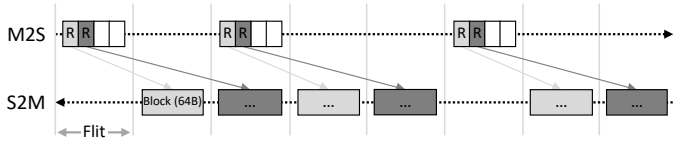
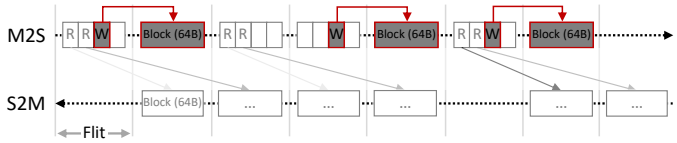


Fig. 2. Bandwidth utilization comparison between DRAM-based Dual In-Line Memory Module (DIMM) and CMM across various read-to-write ratios.



(a) Bandwidth under-utilization when only M2S Read requests are issued.



(b) Improved M2S bandwidth utilization with a balanced R/W ratio.

Fig. 3. M2S and S2M bandwidth utilization under different request patterns.

uses full-duplex communication with separate data buses for read and write operations. These observations, confirmed by other studies [1], [4], underscore the need to adapt request policy, conventionally optimized for DIMM's characteristics, to better align with CMM's full-duplex capabilities.

B. Impact of Read/Write Ratio on Bandwidth Utilization

In Figure 3(a), where only M2S Read requests are issued, the M2S bandwidth is not fully utilized, as the M2S Read requests are not continuously issued but occur sporadically. This results in less efficient packing of requests, leaving some gaps in the M2S bandwidth usage. Meanwhile, the S2M bandwidth becomes a bottleneck when returning the requested data blocks, leading to further delays and under-utilization of the M2S as it waits for the S2M to process the responses. In contrast, Figure 3(b) demonstrates a more balanced request pattern with a different R/W ratio. This balanced approach allows both M2S and S2M bandwidths to be more effectively utilized, as the M2S Write requests help maintain continuous use of the M2S.

C. Read/Write Request Ratio Analysis in Real-World

We analyze the workloads to examine the ratio of R/W requests by using the simulation system described in Table III, utilizing workloads from the SPEC CPU2006 [13] and SPEC CPU2017 [14] benchmark suites. Figure 4 illustrates the read and write request ratios for various workloads. On average, read requests account for approximately 70% of the total requests, while write requests make up around 30%. This highlights the read-dominant nature of the current workloads sent to the CMM. A notable example is the 505-mcf workload, where the read ratio reaches as high as 98%, demonstrating an extreme skew towards read operations. These R/W imbalances suggest that, under the current read-heavy conditions, the full potential of CMM bandwidth utilization is not being fully realized.

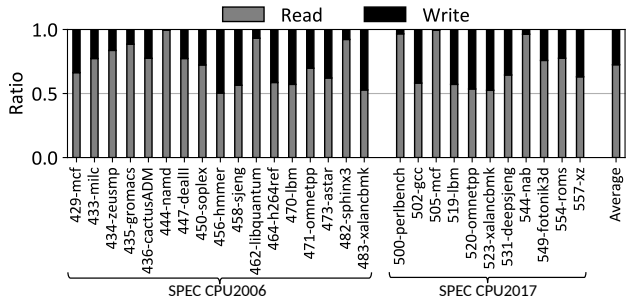


Fig. 4. Ratio of read and write requests across various real-world workloads from SPEC CPU2006 and SPEC CPU2017 benchmarks.

IV. Buddy ECC

The observation of bandwidth under-utilization in CMM and the asymmetric read-to-write ratio request patterns has motivated our approach. Similar to the early write-back schemes proposed in various studies [21]–[23], we proposed Proactively Write-back Policy minimizes the number of dirty cachelines in the last-level cache (LLC), thereby improving bandwidth efficiency. Furthermore, it leverages CMM's capabilities to mitigate the performance degradation typically associated with early write-back. As a result, most cachelines remain in a clean state, allowing for quick access through simple parity checks. This approach also allows the option to implement more robust ECCs, specifically the Buddy ECC we propose to further enhance data reliability in the CMM system.

To enhance data reliability by applying Buddy ECC in CMM memory system, we propose the Proactively Write-back and Utilization-aware Policy, which optimize bandwidth utilization by maintaining mostly clean cachelines, as described in the following subsections.

A. Proactively Write-back

The Proactively Write-back mechanism applied to the CMM uses the varying rewrite distances observed across different workloads to predict blocks that are unlikely to be rewritten [24]. Rewrite distance refers to the number of the LLC accesses that occur before a dirty cache block is written back again. By using rewrite distance, we proactively write-back dirty dead blocks that are unlikely to be rewritten to the CMM, keeping the cachelines mostly clean without repeated write requests.

Figure 5 shows the process of Proactively Write-back. When a write request is issued to the LLC (❶), the rewrite distance history table is updated accordingly (❷). The data for the write request is written to the cache, and the blocks within the set are compared based on their respective rewrite distances (❸). If the rewrite distance for any block exceeds the predefined threshold, the system triggers the Proactively Write-back mechanism to write-back the data to the CMM (❹) and ensuring that mostly all cachelines remain clean state.

B. Utilization-aware Policy

The excessive write requests can overwhelm the internal bandwidth of the CMM, potentially delaying the processing

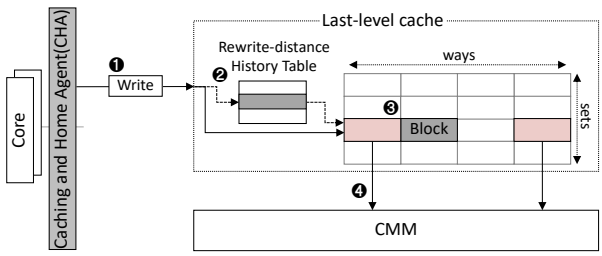


Fig. 5. Proactively Write-back process using rewrite distance.

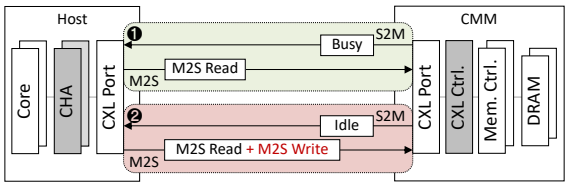


Fig. 6. Dynamic interaction between the host and CMM via M2S and S2M communication channels.

of read requests and resulting in performance degradation. To prevent this from negatively impacting read performance, we implement a strategy that is aware of the internal bandwidth of the CXL-attached DRAM. By monitoring the load on the CMM, we ensure that Proactively Write-back requests are only sent when the CMM is under low-load rate, preventing system overload and ensuring that read requests can still be processed efficiently.

Figure 6 illustrates the data flow between the host and CMM through the interaction of M2S and S2M communication channels. When the CMM sends a "Busy" status to the host via the S2M channel, the host only sends the necessary M2S Read requests without adding additional requests (❶). Conversely, when the CMM transmits an "Idle" status on the S2M channel, the host recognizes that the internal bandwidth of the CMM is not heavily utilized and proceeds to send additional M2S Write requests generated by the Proactively Write-back engine (❷). This adaptive approach minimizes the risk of write-back operations interfering with read performance, optimizing overall system throughput.

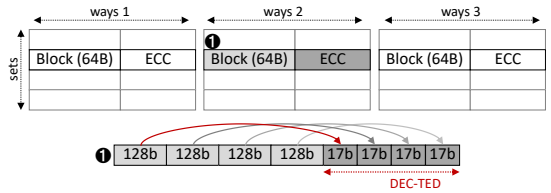
C. Buddy ECC

We can take advantage of the fact that most cachelines are in a clean state due to the Proactively Write-back mechanism, which minimizes the number of dirty cachelines. As demonstrated in various studies, this allows for the separation of the application of Error Detection Code (EDC) and Error Correction Code (ECC) [19], [22]. Usually, EDC is a simple parity code, while modern ECC schemes, such as Double Error Correction-Triple Error Detection (DEC-TED). By applying a simple parity-based EDC to clean cachelines, we free up area that was previously occupied by complex ECC codes. This saved area can then be reallocated to other cachelines, enabling the implementation of a more robust ECC for dirty cachelines.

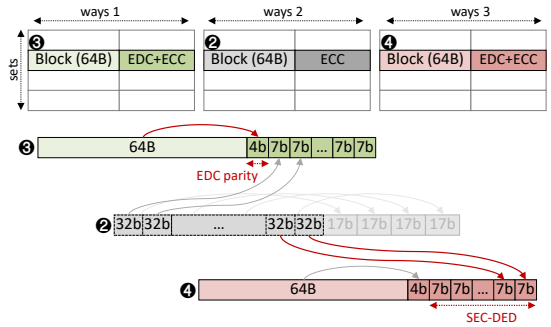
Table I illustrates the overhead of check bits for various data bit (i.e. granularity) sizes when using SEC-DED and DEC-TED ECC schemes [19], [25]. If 64-byte cacheline is protected by four 128-bit granularity DEC-TED codes, this would result in a 68-bit overhead. By separating EDC and ECC, we can implement a simpler EDC using a 1-bit parity per 128-bit clean

TABLE I
ECC AREA OVERHEADS.

	SEC-DED		DEC-TED	
Data bits	Check bits	Overhead	Check bits	Overhead
16	6	38%	11	69%
32	7	22%	13	41%
64	8	13%	15	23%
128	9	7%	17	13%



(a) Conventional cache structure supporting only DEC-TED ECC.



(b) Proposed cache structure with Buddy ECC, providing stronger ECC protection for dirty cachelines.

Fig. 7. Comparison of the conventional DEC-TED supported cache and the proposed Buddy ECC cache structure.

cacheline, freeing up the remaining 64-bit. These saved bits can then be allocated to provide stronger ECC protection for dirty cachelines.

Figure 7 shows the comparison between a conventional cache and a cache utilizing the proposed Buddy ECC. Figure 7(a) illustrates the operation in a conventional cacheline supporting 128-bit granularity DEC-TED . All of 64-byte data block is divided into 128-bit segments, each protected by 17-bit DEC-TED (❶). In Figure 7(b), the dirty cacheline is still protected by DEC-TED in the system using Buddy ECC (❷). If the cacheline is clean, the 64-byte block is protected by a 4-bit simple parity for EDC (❸). To support robust ECC for dirty cachelines, the 32-bit granularity SEC-DED is applied. The 32-bit granularity SEC-DED requires 112-bit, which is stored by utilizing the unused ECC area in the neighboring clean cachelines (❹). The Buddy ECC applies a two-tier ECC mechanism to dirty cachelines. The conventional DEC-TED allows for normal accesses with no additional latency increase, ensuring efficient read operation. When an uncorrectable error is detected through DEC-TED, the fine-grained Buddy ECC activates to correct the error. This tired approach ensures that the system benefits from both low-latency operations under normal conditions and enhanced error correction capabilities when required, without imposing unnecessary performance penalties.

D. Implementation

Figure 8 illustrates the implementation of the Buddy ECC mechanism. When a write request arrives at the LLC, DEC-TED check bits are generated for the corresponding block and stored in its own ECC area (❶). If the adjacent blocks are clean cachelines, additional SEC-DED check bits with higher granularity are generated and distributed across the neighboring clean cachelines (❷). The adjacent blocks refer to the left and

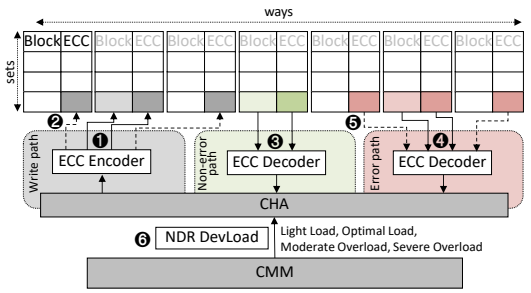


Fig. 8. Implementation of the Buddy ECC system.

right blocks of the cacheline, assuming a simple approach for distributing the SEC-DED check bits. During a read operation on a dirty cacheline, the DEC-TED check bits of the block are used to detect errors. If no-error is detected, the data is returned immediately (❸). However, if a non-correctable error is detected using the DEC-TED check bits (❹), the system checks whether the adjacent cachelines are clean and contain SEC-DED check bits. These bits are read, and used to correct the error (❺). In the case of clean cachelines, if an error is detected, the system performs additional read requests from the lower memory hierarchy to correct the error. Furthermore, as mentioned in Section IV-B, the Utilization-aware Policy uses the *DevLoad* within the Non-Data Response (NDR) defined by the CXL 2.0 specification [15]. This allows the system to communicate the load status of the CMM to the host, effectively preventing CMM overload (❻).

V. Experimental Evaluations

A. Experimental Methodology

We evaluate Buddy ECC using the ramulator2 [12] simulator, which we implement with a CXL-based memory system. A one-way latency overhead of 50ns was added for communication with the CXL system [26]. The experiments are conducted using the environment detailed in Table III, running various benchmarks from SPEC CPU2006 [13] and SPEC CPU2017 [14], all execute on a 4-core setup. Each core undergo a 1-billion instruction warm-up phase followed by the execution of 1-billion instructions. In the graphs present in the following

TABLE II
Real-world System configuration.

Component	Parameter
CPU	Intel®Xeon Family 6 CPUs @2.4 GHz, 56-core
L1 I/D Cache	2.6MB / 3.5MB
L2 Cache	112MB
LLC	288MB
Memory	64GB, DDR5-4800, Bandwidth: 38.4 GB/s
CXL Controller Type	ASIC
Interface Speed	PCIe Express Gen.5 x8, Bandwidth: 31.508 GB/s
Protocol	CXL 2.0
Memory	256GB, DDR5-4800

TABLE III
Simulated system configuration.

Component	Parameter
CPU	4-core, 4-way Out of Order, 3.2GHz
LLC	8MB, 16-way, LRU, 47-cycle
CXL Controller	1.0GHz, One-way latency: 50ns
CMM	DDR5-3200, 32GB, 2-channel, 2-rank tRCD:tCL:tRP:tRAS = 24:24:24:52

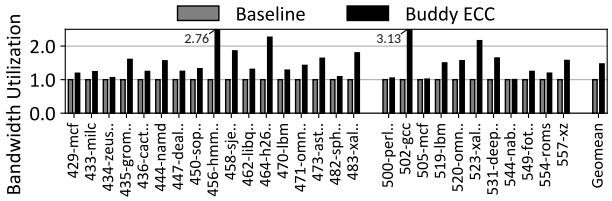


Fig. 9. Bandwidth utilization (normalized to the baseline).

sections, the names on the x-axis (i.e., workloads name) are displayed using abbreviations for simplification.

B. Bandwidth Utilization

Figure 9 illustrates the CXL bandwidth utilization between the baseline system and the system utilizing Buddy ECC across various workloads. On average, Buddy ECC results in a 46% improvement in bandwidth utilization. In particular, two benchmarks 456-hmmer and 502-gcc exhibit the highest increases in bandwidth utilization, with improvements of 176% and 213%, respectively. This significant boost in bandwidth efficiency is attributed to the Proactively Write-back mechanism. As described in Section III-B, the conventional M2S bandwidth is under-utilized. However, with the application of Proactively Write-back, bandwidth optimization is achieved, leading to a more efficient use of available bandwidth.

C. Performance

Figure 10 illustrates the performance comparison between the baseline system and the system utilizing Buddy ECC across various workloads, assuming no errors occur. On average, there is a 0.33% decrease in performance when Buddy ECC is applied, with most workloads showing minimal performance degradation. The 470-lbm workload experiences the most performance decrease, with a 3.6% reduction. However, in some cases, Buddy ECC can actually improve performance. For example, the 482-sphnix3 and 429-mcf workload shows a 2.8% and 2.4% performance increase, respectively attributed to more efficient bandwidth use. By distributing write requests more effectively, read requests are not obstructed, resulting in improved performance. Furthermore, this performance is attributed to the Utilization-aware Policy described in Section IV-B. Consistent with prior studies [27]–[29], the intelligent management of write-back requests demonstrates performance improvements by optimizing bandwidth utilization, thereby contributing to overall system stability.

D. Energy Consumption

Figure 11 shows the energy consumption comparison between the baseline system and the system using Buddy ECC across various workloads. On average, the use of Buddy ECC system results in a 1% increase in energy consumption. The

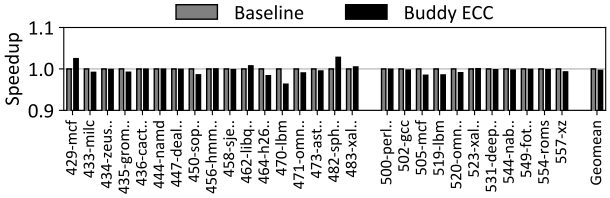


Fig. 10. Performance (normalized to the baseline).

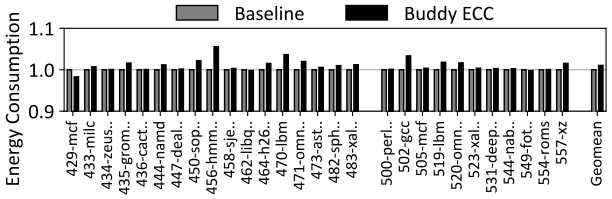


Fig. 11. Energy consumption comparison between the baseline system and Buddy ECC (normalized to the baseline).

456-hmmer workload exhibits the largest increase, with energy usage rising by 5%. In contrast, the 429-mcf workload shows a 1% reduction in energy consumption compared to the baseline. This decrease is attributed to the performance improvement discussed in Section V-C, where the workload completes faster, leading to less overall energy usage. These results indicate that the Proactively Write-back, which leverages rewrite distance as described in Section IV-A, is effectively applied, reducing unnecessary rewrites and contributing to energy savings.

E. Effectiveness of Buddy ECC

Dirty Cacheline Ratio: Figure 12 illustrates the significant reduction in the average number of dirty cachelines per LLC set over time (measured in cycles) in the baseline system and the Buddy ECC system using Proactively Write-back. In the baseline system, an average of 39% of cachelines per set are dirty at any given cycle, whereas the Buddy ECC system reduces this to 8%, representing a 79% reduction in dirty cachelines compared to the baseline. As described in Section IV-C, reducing the ratio of dirty cachelines is crucial for the effective use of Buddy ECC, and the results show that this has been successfully achieved.

Buddy ECC Ratio: Figure 13 shows the ratio of dirty cachelines that can utilize Buddy ECC, measured across cycles. On average, 80% of cachelines are protected by Buddy ECC, highlighting its overall effectiveness across different workloads. The workloads 544-nab and 505-mcf exhibit the highest ratio, with 95% of dirty cachelines being protected by Buddy ECC. In contrast, the workloads 450-soplex and 462-libquantum show the lowest ratio, with only 59% of dirty cachelines protected by Buddy ECC. A key factor driving these results is the proportion of dirty cachelines in each workload. Additionally, by more effectively managing and being aware of the location of dirty

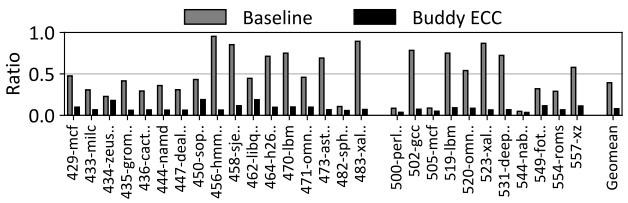


Fig. 12. Dirty cacheline ratio comparison between the baseline system and Buddy ECC.

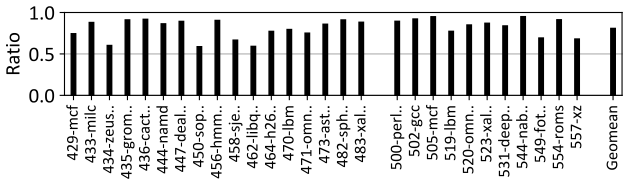


Fig. 13. Ratio of dirty cachelines protected by Buddy ECC.

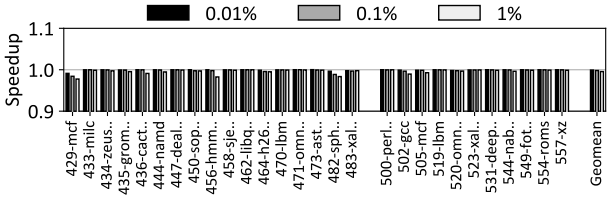


Fig. 14. Performance impact under various error rates corrected by Buddy ECC (normalized to the non-error buddy ECC).

cachelines, it is possible to protect an even greater number of dirty cachelines with Buddy ECC.

Performance with Buddy ECC: Figure 14 shows the performance impact of error rates that cannot be corrected by conventional DEC-TED but are corrected by Buddy ECC. The experiments are conducted with error rates of 0.01%, 0.1%, and 1%, and the results demonstrate that Buddy ECC causes only a slight performance degradation, with an average reduction of 0.07%, 0.18%, and 0.43% for each respective error rate. In contrast, these errors would likely result in system crashes if handled by conventional DEC-TED. Memory-intensive workload such as 429-mcf experience a more noticeable performance impact, but Buddy ECC still manages to correct the errors with lower performance degradation rather than allowing the system to fail. This highlights Buddy ECC's ability to effectively correct errors with minimal performance impact, even for memory-intensive workloads, where conventional error correction would fail.

VI. Conclusion

In this paper, we introduced Buddy ECC, an error correction system tailored for CXL-based memory systems that optimizes bandwidth utilization and enhances data reliability. By incorporating the Proactively Write-back mechanism and Utilization-aware Policy, Buddy ECC reduces the number of dirty cachelines and ensures efficient memory operations, triggering write-backs only under low-load conditions. Additionally, Buddy ECC separates Error Detection Code (EDC) and Error Correction Code (ECC) to apply stronger protection for dirty cachelines, minimizing overhead for clean ones. Evaluated across SPEC CPU2006 and SPEC CPU2017 workloads, Buddy ECC improved bandwidth utilization by 46%, limited performance degradation to 0.33%, and kept energy consumption increase under 1%, making it a highly effective solution for future CXL memory systems. Moreover, Buddy ECC provides stronger ECC protection with minimal performance degradation.

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.II221170, 50%). Seokin Hong is the corresponding author.

REFERENCES

[1] D. D. Sharma, "Compute express link (cxl): Enabling heterogeneous data-centric computing with heterogeneous memory hierarchy," *IEEE Micro*, vol. 43, no. 2, pp. 99–109, 2022.

[2] D. Gouk, M. Kwon, H. Bae, S. Lee, and M. Jung, "Memory pooling with cxl," *IEEE Micro*, vol. 43, no. 2, pp. 48–57, 2023.

[3] D. Das Sharma, R. Blankenship, and D. Berger, "An introduction to the compute express link (cxl) interconnect," *ACM Computing Surveys*, vol. 56, no. 11, pp. 1–37, 2024.

[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, pp. 105–121.

[5] M. Ahn, A. Chang, D. Lee, J. Gim, J. Kim, J. Jung, O. Rebholz, V. Pham, K. Malladi, and Y. S. Ki, "Enabling cxl memory expansion for in-memory database management systems," in *Proceedings of the 18th International Workshop on Data Management on New Hardware*, 2022, pp. 1–5.

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

[7] A. M. Cabrera, A. R. Young, and J. S. Vetter, "Design and analysis of cxl performance models for tightly-coupled heterogeneous computing," in *Proceedings of the 1st International Workshop on Extreme Heterogeneity Solutions*, 2022, pp. 1–6.

[8] B. Tian, Y. Li, L. Jiang, S. Cai, and M. Gao, "Ndpbridge: Enabling cross-bank coordination in near-dram-bank processing architectures," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 628–643.

[9] H. Hassan, "Improving dram performance, reliability, and security by rigorously understanding intrinsic dram operation," *arXiv preprint arXiv:2303.07445*, 2023.

[10] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[11] A. O. Munch, N. Nassif, C. L. Molnar, J. Crop, R. Gammack, C. P. Joshi, G. Zelic, K. Munshi, M. Huang, C. R. Morganti *et al.*, "2.3 emerald rapids: 5th-generation intel® xeon® scalable processors," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 40–42.

[12] 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.

[13] SPEC CPU2006, https://www.spec.org/cpu2006, 2006.

[14] SPEC CPU2017, https://www.spec.org/cpu2017, 2006.

[15] CXL Consortium, "Compute express link 2.0 specification," 2020. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[16] PCI-SIG, "Pci express base specification revision 5.0, version 1.0," 2019.

[17] D. D. Sharma, "Pci-express: Evolution of a ubiquitous load-store interconnect over two decades and the path forward for the next two decades," *IEEE Circuits and Systems Magazine*, vol. 24, no. 2, pp. 47–61, 2024.

[18] O. Kwon, Y. Lee, and S. Hong, "Pinning page structure entries to last-level cache for fast address translation," *IEEE Access*, vol. 10, pp. 114 552–114 565, 2022.

[19] D. H. Yoon and M. Erez, "Memory mapped ecc: Low-cost error protection for last level caches," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 116–127.

[20] 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

[21] J. Hong and S. Kim, "Ecc string: Flexible ecc management for low-cost error protection of l2 caches," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 512–513.

[22] S. Kim, "Reducing area overhead for error-protecting large l2/l3 caches," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 300–310, 2008.

[23] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Soft error and energy consumption interactions: A data cache perspective," in *Proceedings of the 2004 international symposium on Low power electronics and design*, 2004, pp. 132–137.

[24] M. Bakhshalipour, A. Faraji, S. A. V. Ghahani, F. Samandi, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Reducing writebacks through in-cache displacement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 2, pp. 1–21, 2019.

[25] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, 2005.

[26] 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, pp. 574–587.

[27] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens, "Eager writeback-a technique for improving bandwidth utilization," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000, pp. 11–21.

[28] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John, "The virtual write queue: Coordinating dram and last-level cache policies," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 72–82, 2010.

[29] Y. Lee, O. Kwon, and S. Hong, "Don't open row: rethinking row buffer policy for improving performance of non-volatile memories," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 823–828.