LALRU: A Latency-aware LRU Policy for Heterogeneous Memory Systems

Zecheng Li[†], Masayuki Sato[†], Katsuhiko Komatsu[*], Hiroaki Kobayashi[†]
[†]Graduate School of Information Sciences, Tohoku University, Sendai, Japan
[*]Cyberscience Center, Tohoku University, Sendai, Japan
{li.zecheng.t2@dc., masa@, komatsu@, koba@}tohoku.ac.jp

***Abstract*—In data centers, the high cost and limited capacity of DDR main memory have spurred the adoption of heterogeneous memory systems that combine fast DDR with slower and larger-capacity memories. This shift is facilitated by new memory devices and novel interconnect technologies. However, these approaches can reduce the effectiveness of using heterogeneous memory systems due to the significant penalties incurred when fetching cache lines from slower memories with variable latencies. Given the variable latency introduced by heterogeneous memory, there is a need for a nuanced cache replacement policy that accounts for these latencies. Therefore, this paper proposes a latency-aware LRU (LALRU) policy tailored for heterogeneous memory systems. The LALRU policy dynamically adjusts cache line retention by monitoring miss latencies from various memories. Experimental results indicate that the LALRU policy enhances system performance by up to 9.5%, with an average improvement of 4.74% compared with LRU.**

***Index Terms*—Cache replacement policy, heterogeneous memory system, memory disaggregation, cache miss latency.**

I. Introduction

The surge in data-intensive applications has significantly increased memory demand, especially in data centers where DRAM is both costly and limited in capacity [1], [2]. To address this challenge, researchers are exploring alternative memory technologies and heterogeneous memory systems. For example, a DRAM-NVM heterogeneous memory system combines the speed of DRAM with the larger capacity of emerging Non-Volatile Memory (NVM) technologies like PCM [3] and STT-RAM [4].

Furthermore, technologies like CXL [5] will enhance the heterogeneity of memory systems by enabling disaggregated memory connections. The idea of memory disaggregation is to pool memory resources from multiple server nodes, addressing challenges like memory fragmentation due to limited DIMM slots [6], [7], [8] and extend the overall memory capacity. In this context, DRAM main memory directly connected to the CPUs is referred to as "local memory," while NVM or disaggregated memory accessed over interconnects is termed "far memory." These heterogeneous systems introduce a greater diversity of access latencies for far memory, primarily due to the additional delays from interconnect technologies and the inherent characteristics of different memory types.

The variability in latency caused by heterogeneous memory systems further complicates LLC management, rendering previous replacement policies inadequate. In these systems, the Last-Level Cache (LLC) plays a crucial role in mitigating performance impacts by caching data that would otherwise be fetched from slower NVM. However, existing LLC replacement policies that treat all cache lines uniformly may lead to suboptimal performance. Higher hit rates in the LLC do not always guarantee better performance due to the high miss penalties associated with NVM cache lines [9]. Although new methods have been proposed to differentiate between NVM and DRAM cache lines in LLC replacement policies [10], [11], these methods rely on LLC partitioning to manage cache lines from different memory regions, specifically DRAM and NVM regions. However, increasing system heterogeneity reduces the significance of LLC partitioning. Therefore, to maintain optimal performance, a new LLC replacement policy is needed that recognizes and accounts for the latency varieties introduced by these more heterogeneous systems.

In this paper, we propose a Latency-Aware LRU (LALRU) replacement policy that adjusts cache line retention based on real-time latency measurements. LALRU optimizes cache performance for varied workloads by considering different memory access latencies. Experimental results show that LALRU outperforms the state-of-the-art HAP 2-chance policy [10] and the baseline LRU policy.

II. Heterogeneous Memory Systems and their Issues

Heterogeneous memory systems can be categorized into two types: inclusive data access and exclusive flat address. Inclusive systems use a small amount of local DRAM as a new cache layer in main memory to hide long latencies of NVMs [12]. On the other hand, the flat-address exclusive architecture [13], [14], [15] integrates the extended far memory directly into the main memory, thus increasing its overall capacity. This approach not only fully utilizes the spaces of the two memory media, making it more efficient, but also has lower hardware overhead than the inclusive architecture; therefore, it is the main focus of this research. Figure 1 shows an overview of the exclusive heterogeneous memory architecture. This approach can greatly improve the overall memory capacity; however, it also suffers from the far memory latency since LLC can directly access cache lines from both local and far memory, making it critical for overall performance.

To better understand the weakness of the heterogeneous memory system, we use the GEM5 simulator [16] to simulate the system performance of a NVM-DRAM based system with

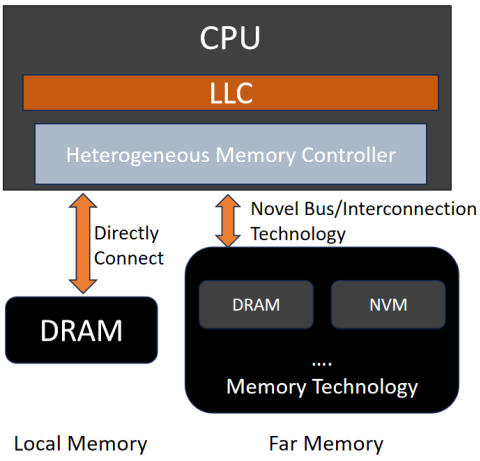

Fig. 1. A Heterogeneous Memory System.

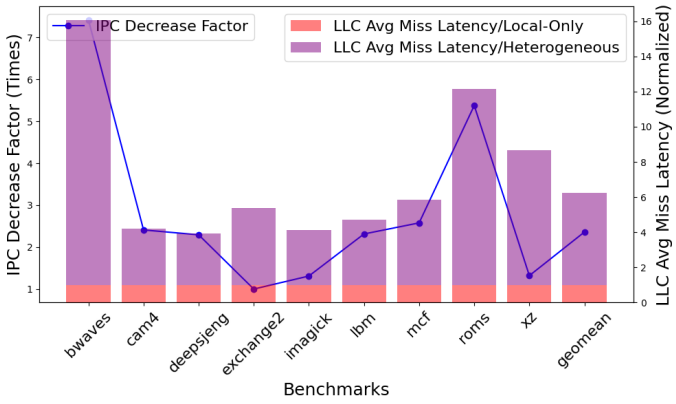

Fig. 2. Comparison between conventional and heterogeneous memory systems.

1:9 ratio of the local and far memory configuration. We set the DRAM access latency to 80 ns, while the NVM latency is 200 ns for both write and read operations to simulate the disaggregated DRAM latencies [8]. Furthermore, we evaluate IPC performance when executing one billion instructions test under SPEC CPU 2017.

Figure 2 shows the overall IPC performance. The results show that an average IPC drops by four times compared with the DRAM-only system. Figure 2 also shows the results of the LLC miss latency. We noticed that one of the most significant discrepancies is found in the average LLC miss latency. The average miss latency increased more than six times compared with the DRAM-only system. This is because the LLC directly fetches cache lines from both local and far memory regions. Therefore, this paper concludes that the miss penalty of LLC is one of the key bottlenecks directly affecting overall performance.

Some researchers have been focused on improving the performance by better utilizing local memory. For example, data management strategies including page migration are employed [17], [18]. These strategies reallocate ”HOT” frequently-accessed pages to local memory and ”COLD” rarely-used pages to far memory in DRAM-NVM based heterogeneous memory systems, aiming at improving both local memory utilization and system performance. Recent studies are exploring this technique in disaggregated memory contexts [8], [19]. However, the capacity of local memory remains limited and depends on the number of Dual In-line Memory Modules (DIMM) interfaces. Therefore, even with continuous page migration redirecting pages from far memory to local memory, it is unavoidable for the CPU to directly access cache lines from far memory. Consequently, the LLC may simultaneously fetches cache lines from both local and far memory.

This also raises a demand for better utilization of the heterogeneous memory systems, especially in the aspect of reducing long LLC miss penalty. Several studies have been done to enhance LLC replacement policies for DRAM-NVM based heterogeneous memory systems [10], [11]. However, they mostly focus on NVM memory characteristics such as their high writeback latency.

The Writeback-aware Dynamic Cache (WADE) [11] strategy manages the LLC by partitioning it into lists based on the writeback frequency. This approach reduces writeback requests by retaining frequently utilized dirty blocks within the LLC, improving both the cache hit rate and reducing data requests to the NVM main memory.

The Hybrid-memory-aware Partition (HAP) [10] dynamically segments the Last Level Cache (LLC) into distinct partitions for NVM and DRAM based on memory request patterns, enhancing the management of dirty data. It employs a modified LRU policy, named HAP 2-chance, which incorporates a hit counter. Specifically, if a far memory cache line is slated to be evicted in the LRU has previously been hit during its current lifetime in the LLC, it is reinserted at the highest priority position of the LRU priority list, thereby receiving a second chance to remain in the LLC.

However, the applicability of these methods becomes difficult. In a heterogeneous memory system that relies on various memory sources, partition-based strategies need a lot of partitions. The management of these partitions becomes complicated if the number of memory regions increases. Our primary insight is that increasing system heterogeneity reduces the significance of LLC partitioning. As memory resources introduce latency variability, planning effective LLC partitions and implementing optimal replacement policies become challenging. Therefore, the system needs to improve its ability to identify and respond to varying miss penalties in heterogeneous memory systems, as well as track cache line attributes, in order to make better decisions about managing cache lines in the LLC and improve performance.

III. A Latency-aware Cache Replacement Policy for Heterogeneous Memory Systems

A. Basic Concept

To optimize cache performance in heterogeneous memory systems, this paper proposes a Latency-aware LRU replace-

TABLE I
EXAMPLE TO COMPARE LRU AND LALRU.

LRU Priority	8	7	6	5	4	3	2	1
Cache line	H	G	F	E	D	C	B	A
Miss Latency (ns)	80	80	80	80	80	150	80	400

ment policy (LALRU). The core concept of LALRU is to enhance the traditional Least Recently Used (LRU) cache replacement policy by incorporating awareness of various cache line latencies. The traditional LRU does not account for these differences, potentially leading to inappropriate caching decisions. LALRU addresses this by adjusting the eviction order of cache lines based on their miss penalties. Cache lines with higher miss penalties, which are typically from far memory, are prioritized for retention in the cache. On the other hand, cache lines with lower miss penalties are more likely to be evicted. This approach reduces the overall memory access latency by minimizing the likelihood of costly cache misses for high-penalty cache lines.

B. Prioritizing Far Memory Cache Lines in LLC Retention

To adjust the eviction order of cache lines, LALRU cancels the evictions of far memory cache lines and inserts them to the higher priority positions in the LRU priority list. For an insertion to occur, two conditions must be met: first, the cache line about to be evicted must have a latency greater than the combined latency of one or more preceding cache lines in the LRU priority list; however, if the preceding cache line in the LRU priority list has a higher latency than the cache line being evicted, the insertion will not be triggered. Second, the insertion by LALRU should occur only for the cache line that receives at least one hit. This condition ensures that LALRU prevents cache lines with the low possibility of being reused from occupying the LLC space indefinitely.

Regarding the insertion position, LALRU does not simply insert the far memory cache line to be evicted back to the top of the LRU priority list like HAP 2-chance [10]. Instead, within the priority list provided by LRU, LALRU calculates the maximum tolerable penalty X for the cache line. This maximum tolerable penalty X determines the insertion position, which is the X-th position in the priority list. The formula for calculating the maximum tolerable penalty X is as follows:

$$X = \max\left\{ i \mid \sum_{j=2}^{i} l_j < P \right\} \tag{1}$$

Here, $L = [l_1, l_2, \ldots, l_n]$ is the priority list, where each l_j represents the latency of the j-th priority cache line. P is the miss latency of the cache line that is to be evicted. $\sum_{j=2}^{i} l_j$ represents the sum of the latencies of the lower-priority i cache lines in the priority list, excluding the one to be evicted. Consequently, this formula finds the largest integer i such that the sum of the latencies from the 2nd to i-th priority cache lines does not exceed P and signs this value to X.

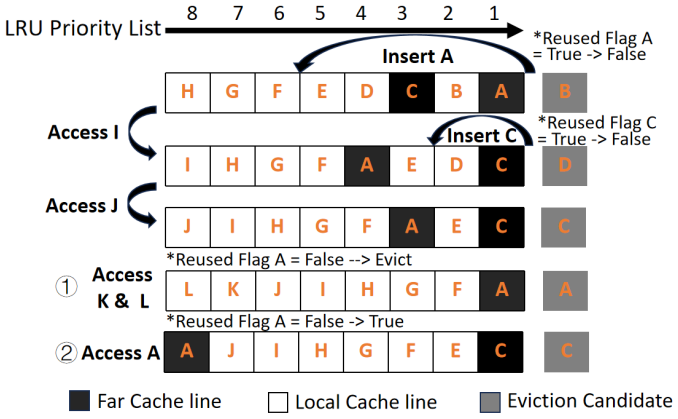

Fig. 3. Example of LALRU Behavior.

C. Examples

To show the effectiveness of the insertion based on the maximum tolerable penalty, Table I shows an example of a 8-way cache. The LRU priority represents the eviction priority of each cache line. In this example, cache lines A and C are from the far memory region with a miss latency of 400 ns and 150 ns respectively. When a new cache line I is stored into the cache, it triggers an eviction of cache line A. In the worst case, the next operation requires fetching cache line A again, resulting in a cache miss with a latency of 400 ns. However, if preceding cache lines B, C, D, and E are evicted instead, the situation improves. Even in the worst case where all these cache lines are fetched immediately after eviction, the total miss latency would be 390 ns. This is still lower than the 400 ns latency caused by directly evicting cache line A. In this case, the ideal insertion position for cache line A is the 4-th priority position that can be calculated by the maximum tolerable penalty, and cache lines D, C, and B should be moved to the right-hand side one by one. After that, a new cache line is stored into the LLC and cache line B is evicted.

LALRU can give each cache line an ideal life time in the LLC. To demonstrate that, Figure 3 shows another long-run example of LALRU. After accessing new cache line I, the insertion of cache line A is triggered, and cache line B is evicted as discussed in the previous paragraph. Next, an access to new cache line J occurs, far cache line C will also triggers an insertion according to its maximum tolerable penalty, and inserted behind the cache line D. As this figure depicts, two cache lines can have different LLC life times along with their latencies.

Moreover, LALRU can apply insertions only for cache lines with the high possibilities to be reused and avoid insertions of cache lines with the low possibilities. To think these cases, Figure 3 shows the two upcoming scenarios after the two insertions.

- Scenario ①: After new cache lines H to L are accessed and if cache line A did not get access in the upcoming life cycle inside the LLC, cache line A keeps its reused flag unset and comes to the 1st prioirty position of the

Algorithm 1: LALRU Algorithm

```
Data: candidates, N
1  L ← find N cache lines in the set
2  Sort L by tick in ascending order
3  if L[1].reusedFlag is unset then
4      return L[1]
5  else if L[2].missPenalty > L[1].missPenalty then
6      return L[1]
7  Sum ← 0
8  for X = 2 to N do
9      Sum ← Sum + L[X].missPenalty
10     if Sum > S1.missPenalty then
11         Break
12 L[1].reusedFlag ← false
13 L[1].tick ← L[X].tick
14 for i = 2 to X − 1 do
15     L[i + 1].tick ← L[i].tick
16 return L[2]
```

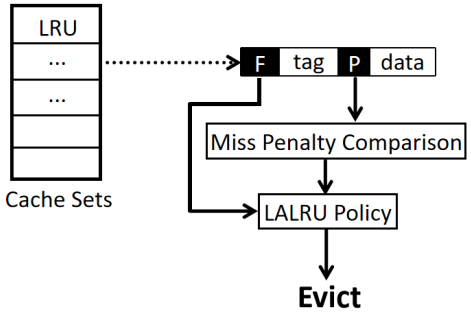

Fig. 4. LALRU Hardware.

LRU priority list again. If one more new cache line comes into the cache, cache line A will be directly evicted.

- Scenario ②: If the cache line A get hits in the upcoming life cycles inside LLC, it will be inserted to the 8th priority position of the LRU priority list just like LRU did. Then, its reuse flag is set, meaning that it has another chance to be inserted by LALRU when it comes to the 1st priority position of the LRU priority list.

D. LALRU Implementation

Algorithm 1 shows the overall procedure of LALRU implementation written in pseduo code. We use clock ticks from a global simulator clock to track the LRU priority list. First, the first candidate with the smallest ticks in the LLC (i.e., the first priority cache line in the LRU priority list) is checked for its reused flag in Lines 3 to 4. Once they meet the insertion requirements from Section III-B, we use a loop that starts at Line 8 and sums up the latencies of cache lines to obtain the maximum tolerable penalty X. Their ticks are adjusted accordingly to update their positions in the LRU priority list.

Figure 4 shows the hardware overview of LALRU. LALRU adds two parameters to the baseline LRU. First, it tracks the miss latencies of cache lines and stores the record of the latencies when they enter the LLC, which are the extra fields named P. To reduce the hardware storage overhead for recording the miss latencies, we categorize the latencies into multiple levels. In this paper, additional 3 bits to track 5 different levels are assigned to each cache line. Second, a reused flag F of each cache line is used to indicate whether the cache line is reused after the insertion by LALRU or not. These flags consume one flag bit for each cache line. In summary, we extend the LRU design with extra 4 bits per cache line in total while keeping the time complexity the same as the LRU policy.

TABLE II
SIMULATION PARAMETERS.

Object	Specification
Cores	Trace CPU at 2 GHz, 1 (ST), 4(MT)
L1 Caches	32 KB, 8-way, split D/I, 1-cycle
L2 Caches	Private, 256 KB, 8-way, inclusive, 7-cycle
L3 Cache	Shared, 1MB(ST)-4MB(MT), 16-way, inclusive, 27-cycle
Local Memory	DDR3, 80 ns Latency
Far Memory	NVM with Similar Write and Read Latency, 200ns, 240ns, 320ns, 400ns, 480ns

TABLE III
CLASSIFICATION BY MPKI FOR SINGLE AND MULTI-THREAD BENCHMARKS.

Single Thread	
MPKI	Benchmarks
Low MPKI (<1)	exchange2, imagick, xz
Medium MPKI (1∼10)	lbm, cam4, bwaves
High MPKI (>10)	roms, mcf, deepsjeng
Multi-thread (4 Threads)	
MIX1: exchange2, imagick, xz, lbm	
MIX2: imagick, xz, lbm, cam4	
MIX3: xz, lbm, cam4, bwaves	
MIX4: lbm, cam4, bwaves, roms	
MIX5: cam4, bwaves, roms, mcf	
MIX6: bwaves, roms, mcf, deepsjeng	

IV. EVALUATION

A. Experimental Setups

To evaluate the proposed cache management policy for the heterogeneous memory systems, the simulation experiment is conducted. The simulator is developed based on GEM5 [16] that can simulate the hardware architecture that includes processors, a cache hierarchy, and an heterogeneous memory system. Table II shows main configuration parameters of the simulated system. We configured the local memory to have an access latency of 80 ns. Additionally, the far memory is set with multiple read access latencies ranging from 200 ns to 480 ns and similar write latencies.

To evaluate the performance of the heterogeneous memory system, nine benchmarks shown in Figure III are selected based on different MPKI values. For measuring the performance, each benchmark is executed with 1 billion instructions. In addition, to keep the memory foot print of the heterogeneous memory system fixed, we do not apply any page

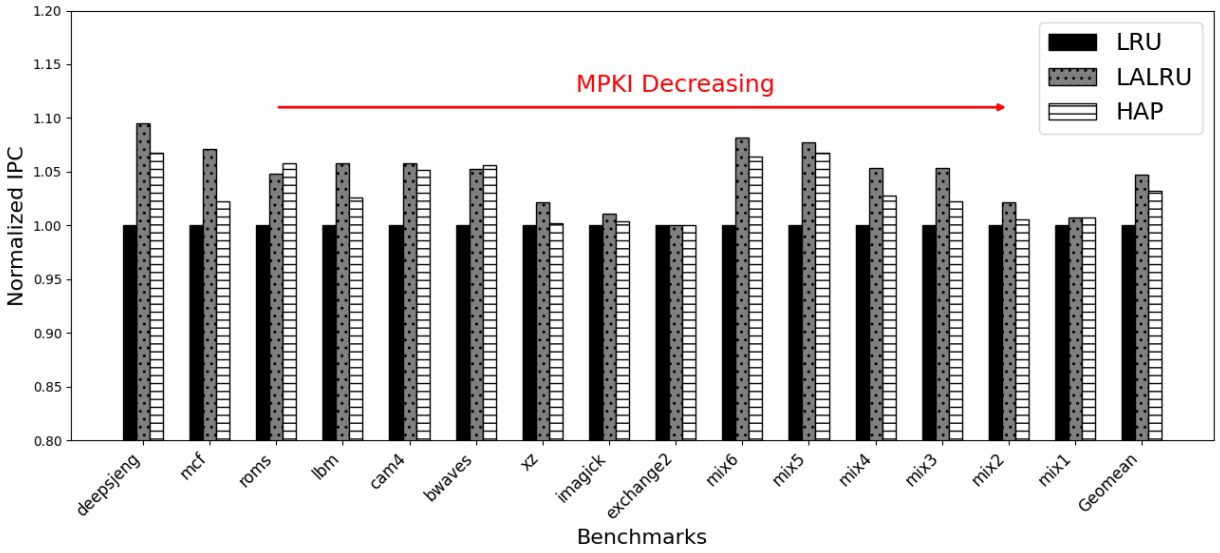

Fig. 5. IPC Performance.

migration operation and strictly keep the usage of local and far memory fixed. The default usage ratio of the local memory to the far memory is set to 1:5.

Meanwhile, we compare HAP [10] and LRU against LALRU. We pointed out that the LLC partition is no longer effective for heterogeneous memory systems. Therefore, the evaluation of HAP ignores the partition mechanism but only focuses on the HAP 2-chance replacement policy.

B. IPC Evaluation

Figure 5 shows the evaluation results. The x-axis represents the benchmarks, and the y-axis represents the normalized IPC values of each replacement policy relative to LRU. Figure 5 indicates that the LALRU replacement policy significantly improves its IPCs for most benchmarks. Notably, in high MPKI scenarios, the single-threaded cases show substantial improvements. For example, $deepsjeng$ achieves a 9.5% improvement over LRU. This is likely because, in high MPKI applications, cache lines in the LLC are frequently replaced. The LALRU strategy effectively prioritizes retaining far memory cache lines, resulting in greater overall benefits, with an average improvement of 4.74%.

Meanwhile, LALRU outperforms HAP 2-chance due to the aggressive insertion strategy of HAP 2-chance since it only inserts eviction candidates back to the top of the LRU priority list, which gives the cache line from far memory inappropriate lifetime in LLC and eventually affects the overall performance.

Figure 5 also indicates that, in benchmarks with a low MPKI, the performance of LALRU remains mostly unchanged. For instance, $exchange2$ has an extremely low MPKI, approximately 0.01. This means that the LLC efficiently utilizes the principle of locality and implies a high hit rate, reducing the number of accesses to the main memory.

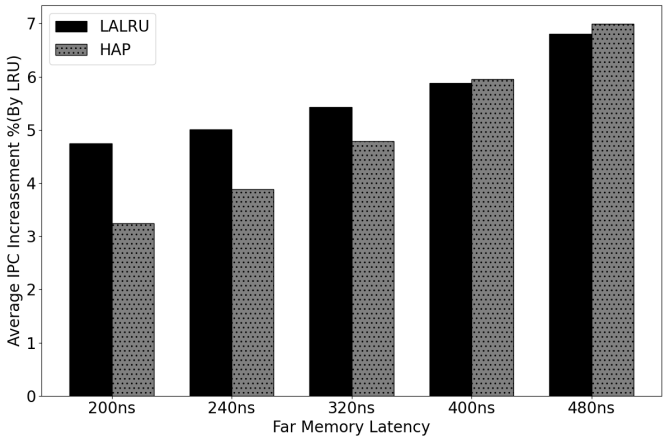

Fig. 6. Sensitivity to Latency.

C. Latency Sensitivity

Although LALRU demonstrates better performance than HAP 2-chance in Section IV-B, further simulation experiments with varying far memory latencies are conducted for a more comprehensive evaluation. Figure 6 shows the evaluation results. The x-axis represents different far memory latencies listed in Table II. The y-axis shows the average IPC improvement against LRU across all the benchmarks we selected.

From Figure 6, it can be observed that, when the far memory latency is below 400ns, LALRU still outperforms HAP 2-chance. This is likely due to the insertion strategy of LALRU, which is more conservative than that of HAP and helps maintain the LLC hit rate. However, as the latency exceeds 400ns, HAP 2-chance starts to show better improvements, eventually surpassing LALRU. This shift is because, as the far memory latency arises, the benefits of prioritizing the far memory cache lines outweigh the advantages of maintaining

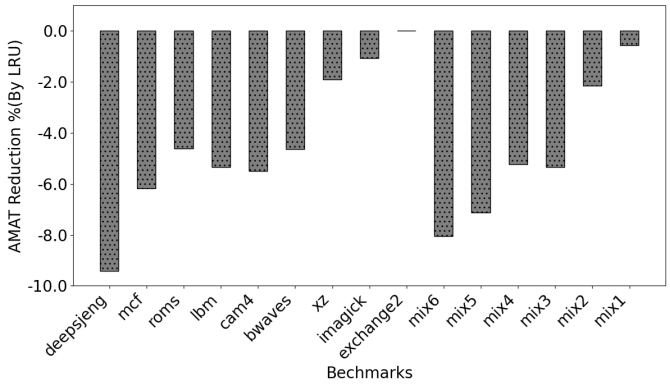

Fig. 7. AMAT Reduction.

the overall LLC hit rate.

In summary, the evaluation results show that LALRU is more effective at lower latencies due to its conservative strategy. On the other hand, HAP 2-chance becomes superior as the latency increases over 400ns. This demonstrates the importance of tailoring cache replacement policies to specific memory latency conditions to optimize the performance of heterogeneous memory systems.

D. Average Memory Accessing Time

To further evaluate the effectiveness of the proposed policy, Average Memory Access Time (AMAT) is evaluated for each benchmark. The AMAT formula combines several critical factors, including cache hit time, miss rate, and the penalty for misses. These factors together create a comprehensive metric that is crucial for evaluating the total latency reduction in LALRU. The formula of AMAT is shown below:

$$\text{AMAT} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty} \quad (2)$$

LALRU aims to optimize AMAT by reducing the frequency of long-latency memory accesses. This approach means that even though LALRU slightly disrupts the eviction order of LRU, the overall memory access time might still be optimized due to its fewer long-latency accesses. Thus, AMAT becomes an intuitive and effective metric for assessing improvements by LRU for heterogeneous memory systems. By measuring AMAT, we can clearly demonstrate whether LALRU effectively reduces memory access delays in practical applications.

The x-axis of the Figure 7 represents the benchmarks, while the y-axis shows the reduction in AMAT compared with LRU. The latency of the far memory is set to 200 ns in this experiment. The results indicate that LALRU significantly reduces the AMAT compared with LRU. AMAT is reduced by an average of 4.48%, aligning well with our expectations of the LALRU performance.

This detailed analysis demonstrates that LALRU can lead to a substantial reduction in AMAT. The consistency of these results across different benchmarks underscores the potential of LALRU to optimize memory access times significantly. The evaluation results of AMAT not only support the initial findings from the IPC analysis but also provide a more nuanced understanding regarding the impact of LALRU on system performance.

V. Conclusions and Future Work

The paper proposes LALRU, a novel replacement policy for heterogeneous memory systems. LALRU prioritizes far memory cache lines for retention in the LLC to reduce overall access latency. Experimental results indicate that LALRU performs well across most benchmarks, particularly showing a 9.5% improvement in the single-threaded $deepsjeng$ benchmark over traditional LRU, with an average improvement of 4.74%. It effectively lowers the average memory access time (AMAT) by extending the lifecycle of far memory cache lines, with an average reduction of 4.48%, outperforming HAP 2-chance.

Future work should further explore the applicability of LALRU in other systems, particularly its adaptability and optimization in environments with OS-level page migration capabilities. Additionally, the paper suggests that LALRU could also enhance replacement policies in DRAM caches and storage disk buffers, where the access latency is more pronounced.

References

1 Xue, C. J., Zhang, Y., Chen, Y., Sun, G., Yang, J. J., and Li, H., "Emerging non-volatile memories: opportunities and challenges," ser. CODES+ISSS '11, 2011.

2 Li, D., Vetter, J. S., Marin, G., McCurdy, C., Cira, C., Liu, Z., and Yu, W., "Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications," ser. IPDPS '12, 2012.

3 Nirschl, T., Phipp, J. B., Happ, T. D., Burr, G. W., Rajendran, B., Lee, M.-H., Schrott, A., Yang, M., Breitwisch, M., Chen, C.-F., Joseph, E., Lamorey, M., Cheek, R., Chen, S.-H., Zaidi, S., Raoux, S., Chen, Y. C., Zhu, Y., Bergmann, R., Lung, H.-L., and Lam, C., "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Proceedings of the Electron Devices Meeting*, 2007.

4 Dieny, B., Sousa, R., Bandiera, S., Souza, M., Auffret, S., Rodmacq, B., Nozieres, J. P., Herault, J., Gapihan, E., Prejbeanu, I. L., Portemont, C., Mackay, K., and Camb, B., "Extended scalability and functionalities of mram based on thermally assisted writing," in *Proceedings of the International Electron Devices Meeting*, 2011.

5 "Cxl 3.0 specification," https://www.computeexpresslink.org/download-the-specification.

6 Calciu, I., Imran, M. T., Puddu, I., Kashyap, S., Al Maruf, H., Mutlu, O., and Kolli, A., "Rethinking software runtimes for disaggregated memory," in *ASPLOS '21*.

7 Li, H., Berger, D. S., Hsu, L., Ernst, D., Zardoshti, P., Novakovic, S., Shah, M., Rajadnya, S., Lee, S., Agarwal, I., Hill, M. D., Fontoura, M., and Bianchini, R., "Pond: Cxl-based memory pooling systems for cloud platforms," ser. ASPLOS 2023.

8 et al., A. M., "Tpp: Transparent page placement for cxl-enabled tiered memory," *arXiv*, vol. 2206.02878, 2022.

9 Qureshi, M. K. and Loh, G. H., "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," ser. MICRO '12.

10 Wei, W., Jiang, D., Xiong, J., and Chen, M., "Hap: Hybrid-memory-aware partition in shared last-level cache," *TACO '17*.

11 Wang, Z. *et al.*, "Wade: Writeback-aware dynamic cache management for nvm-based main memory system," *TACO '13*.

12 Lee, H. G., Baek, S., Nicopoulos, C., and Kim, J., "An energy- and performance-aware dram cache architecture for hybrid dram/pcm main memory systems," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*.

13 Lee, S., Bahn, H., and Noh, S. H., "Characterizing memory write references for efficient management of hybrid pcm and dram memory," in *Proceedings of the Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2011.
14 Mutlu, O., Ausavarungnirun, R., Harding, R. A., Meza, J., and Yoon, H., "Row buffer locality aware caching policies for hybrid memories," in *Proceedings of the International Conference on Computer Design*, 2012.
15 Ramos, L. E., Gorbatov, E., and Bianchini, R., "Page placement in hybrid memory systems," in *Proceedings of the International Conference on Supercomputing*, 2011, pp. 85–95.
16 Lowe-Power, J. and et al., "The gem5 simulator: Version 20.0+," 2020.
17 Bock, S., Childers, B. R., Melhem, R., and Mossé, D., "Concurrent page migration for mobile systems with os-managed hybrid memory," ser. CF '14, 2014.
18 Agarwal, N. and Wenisch, T. F., "Thermostat: Application-transparent page management for two-tiered main memory," ser. ASPLOS '17, 2017.
19 Patke, A., Qiu, H., Jha, S., Venugopal, S., Gazzetti, M., Pinto, C., Kalbarczyk, Z., and Iyer, R., "Evaluating hardware memory disaggregation under delay and contention," in *IPDPSW '22*.