ReMAP: Reuse and Memory Access Cost Aware Eviction Policy for Last Level Cache Management

Akhil Arunkumar and Carole-Jean Wu
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ 85281, USA
{Akhil.Arunkumar, Carole-Jean.Wu}@asu.edu

Abstract—To mitigate the significant main memory access latency in modern chip multiprocessors, multi-level on-chip caches are used to bridge the gap by retaining frequently used data closer to the processor cores. Such dependence on the last-level cache (LLC) has motivated numerous innovations in cache management schemes. However, most prior works focus their efforts on optimizing cache miss counts experienced by applications, irrespective of the interactions between the LLC and other components in the memory hierarchy such as the main memory. This results in sub-optimal performance improvements, since reducing miss rates does not directly translate to increased IPC performance.

In this paper, we show that in addition to the recency information provided by the cache replacement policy, post eviction reuse distance (PERD) and main memory access latency cost are useful to make better-informed eviction decisions at the LLC. We propose ReMAP, Reuse and Memory Access Cost aware eviction policy, that takes reuse characteristics and memory access behavior into consideration when making eviction decisions. ReMAP achieves higher performance compared to prior works. Our full-system simulation results show that ReMAP reduces the number of misses of SPEC2006 applications by as much as 13% over the baseline LRU replacement and by an average of 6.5% while MLP-aware replacement and DRRIP reduce the miss counts by -0.7% and 5% respectively. More importantly, ReMAP achieves an average of 4.6% IPC performance gain across the SPEC2006 applications while MLP-aware replacement and DRRIP see only 1.8% and 2.3% respectively.

I. INTRODUCTION

In modern computing systems, DRAM access latency is the most significant bottleneck hindering performance due to the widening gap between the speed of the main memory and the processor core. Multi-level on-chip SRAM caches are used to bridge the gap by storing frequently-used data closer to the core. Across the memory hierarchy, last level caches (LLCs) play a crucial role in avoiding fetching data from the main memory, and therefore it is important to effectively manage the performance of the LLC.

The dependence on LLC to support the growth of next generation chip multiprocessors has resulted in numerous innovations in LLC management [1], [2], [5], [6], [8], [12], [15], [18], [20], [24], [26], [31]. Most previous works focus their efforts on optimizing various aspects of LLC performance, e.g., reducing miss rate [5], [14], [24] or guaranteeing Quality-of-Service (QoS) [11], [23], [30]. These cache management policies use variations of recency information of cache lines to determine the cache line eviction/replacement policy. However, these policies often optimize the performance of the LLC in isolation without considering its interaction with the upper level caches, or the main memory. This results in sub-optimal performance improvements since reducing miss rates does not directly translate to increased Instruction-per-Cycle (IPC). In fact, our studies show that such policies leave ample room for improvement. In this paper, we show that additional performance improvement can be achieved by taking the LLC interaction with main memory into account while making cache management decisions.

In the open page DRAM architectures, the cost of a memory access can vary by a large amount between 30 and 300 cycles, depending on whether the memory access experiences a row buffer hit or a bank conflict. Therefore, not all LLC misses experience the same memory access cost. This variation in the memory access latency presents us with an opportunity to optimize the LLC management scheme such as eviction or replacement policy by taking the varying memory access cost into consideration.

Qureshi et al. [25] highlighted the potential of coordinating LLC management with DRAM access behavior. Their proposal, MLP-aware cache management, uses the varying memory access behavior to guide cache line replacement decisions to improve system performance. For example, if the data in a particular cache line would experience a longer memory access latency, the cache line is less likely to be evicted from the LLC. However, this design does not take into account the reuse distance information for the cache line. If a cache line is not going to be reused in the near future, even though it has a longer memory access latency, such a cache line should be replaced by data that will be reused by the processor to better utilize the capacity of the LLC. Through our study, we observe that the likelihood of reuse after a line is evicted and the corresponding reuse distance (Post Eviction Reuse Distance) provides information that can be utilized to make better-informed eviction decisions at the LLC.

We propose ReMAP, Reuse and Memory Access Cost aware eviction policy, that takes cache line reuse characteristics and memory access behavior into consideration when making cache line eviction decision. ReMAP combines recency, post eviction reuse distance, and memory access cost information together to achieve higher performance compared to prior works. We compare ReMAP with two closely-related state-of-the-art prior works, RRIP [14] and MLP-aware replacement [25]. Our full-system simulation results show that ReMAP reduces the number of misses of SPEC2006 applications by as much as 13% over the baseline LRU replacement and by an average of 6.5% while MLP-aware replacement and DRRIP reduce the miss counts by -0.7% and 5% respectively. More importantly, ReMAP achieves an average of 4.6% IPC performance gain across the SPEC2006 applications while MLP-aware replacement and DRRIP see only 1.8% and 2.3% respec-
Our paper makes the following contributions:

1) We identify the limitations of prior-art LLC management schemes that do not take into account cross-hierarchy memory subsystem interactions, such as non-uniform memory access time due to bank conflict and row buffer access characteristics.

2) We develop a simple yet effective cross-hierarchy memory subsystem model to estimate the overall access latency to the memory and use this estimation to guide LLC replacement.

3) We devise a novel LLC management scheme that determines the lifetime of cache lines using cross-hierarchy information, including cache line reuse distance, recency, and memory access cost.

II. BACKGROUND AND MOTIVATION

There has been a wide body of research literature that is directed towards improving cache management. This has led to many innovations in insertion, promotion and replacement policies, and dead block prediction [1], [2], [5], [6], [12], [15], [18], [20], [24], [31]. Most of the aforementioned studies focus on optimizing the LLC performance in isolation. Though the improvement in LLC performance leads to a significant reduction in the gap between memory and processor speeds, LLC is going to be most effective when its working is well coordinated with the levels of memory above and below it, i.e., the L1 and L2 private caches and the DRAM.

In the widely prevalent open page DRAM designs, not all LLC misses experience a fixed memory access cost. The memory access cost can vary from approximately 15ns to 150ns (for a 2GHz processor, this corresponds to 30 cycles to 300 cycles). This is owing to the fact that LLC misses could result in row buffer hits, row buffer misses, or map to conflicting banks in the memory. To address this, Qureshi et al. [25] made a compelling case for taking memory level parallelism into consideration while making the LLC eviction decision. The memory access cost is lower for parallel LLC misses because the cost is amortized over multiple misses.

LLC misses that incur the least memory access cost are the ones that 1) do not cause bank conflict and 2) hit in the row buffer. The memory access cost incurred by such a reference is proportional to the time taken to place the data on the data lines from the sense amplifiers. This is called Column Address Strobe (CAS) latency (CL). Typically CL is about 15ns for a DDR3 SDRAM [7]. Therefore, for a 2GHz processor, the overall memory cost to fetch data from the memory is

\[
\text{MemoryCost}_{\text{row buffer hit}} \propto CL \approx 15\text{ns} = 30\text{ cycles}
\]  

However, when the row buffer of the bank does not contain the row of data for the referenced address, a row buffer miss is incurred. In this case, when the data request is presented to the memory, the row that is open in the row buffer is closed (row is precharged) and the row corresponding to the new reference is brought to the row buffer. Finally, the data is placed on the data lines. The time taken to bring data into the row buffer is referred to as Row Address Strobe (RAS) to Column Address Strobe (CAS) delay. In this case, the overall memory access cost becomes the sum of the time taken for row precharge (tRP), time to bring data into row buffer (RAS to CAS delay or tRCD), and CAS latency. Typically for a DDR3 SDRAM, tRCD and tRP are about 15ns [7]. Therefore for a 2GHz processor, the overall cost to fetch data from the memory is

\[
\text{MemoryCost}_{\text{row buffer miss}} \propto tRP + tRCD + CL \approx 15\text{ns} + 15\text{ns} + 15\text{ns} = 45\text{ns} = 90\text{ cycles}
\]  

When an LLC miss is mapped to a conflicting bank, the memory access cost experienced varies (depending on the memory access costs of earlier misses that are waiting to be serviced by this particular bank), and a cascading effect influences the memory access cost. If two requests map to a bank when another bank is idle, the second request experiences a memory access cost that is the sum of the memory access cost experienced by the first access and the second request’s own memory access cost. If both the first and second requests experience a row buffer miss, this could be as high as 180 cycles (90 cycles for the first request and 90 cycles for this request).

Next we use our characterization results to highlight that we can choose the eviction candidates in the LLC more intelligently, if the knowledge of the memory access cost and reuse pattern is available at the time of cache line replacement. For various SPEC2006 benchmarks, Figure 1 shows the memory access cost breakdown of the lines that are evicted from a LRU-based LLC. Utilizing oracle information, we also show the fraction of evicted lines that would have been reused in the future. We notice that **live cache lines are being evicted from the LLC while there are one or more other cache lines in the same set, that are dead.** Among all evicted cache lines, the fraction of cache lines that are indeed dead (with no future or distant reuse) is represented by the black bars labeled “Dead Lines” in Figure 1. The adjacent grey bars show the fraction of evictions when there is at least one cache line in the same set, that is dead. The difference between these two bars gives the fraction of times where a live line was evicted even though a dead line was available in the cache. We can see that this difference is significant for many benchmarks – 15% on an average and as much as 50%, of all evictions. This is because using only recency information is ineffective in identifying the best cache eviction candidates.

While rescuing cache lines that are still useful in the near future can improve the LLC performance, we observe that the memory access penalties for these cache lines can vary significantly. For example, some cache lines that could be reused in the near future have longer memory access cost than others. This leads to our second important observation: **cache lines being evicted from the LLC are not always the ones that have the least memory access cost.** Often there are one or more cache lines in the same set whose memory access cost is lesser than memory access cost of the chosen eviction candidate. The darker bars in Figure 1 represent the fraction of evictions where live lines with higher memory access cost were evicted. This undesirable behavior occurs to 10% of the evictions on average (and can happen to as much as 60% of the evictions). For most of these occasions, there is opportunity to convert higher memory access cost evictions to dead line evictions or lower memory access cost evictions. Therefore, we need a cache replacement policy that prioritizes cache lines with longer memory access cost and farther or no reuse over other lines in the set at the eviction time.

From the above insights, we can see that standard recency-based cache replacement policies leave sufficient
room for improvement. Leveraging on post eviction reuse distance (PERD) and memory access cost (MAC) information along with recency information can provide additional performance benefits. In this paper, we propose ReMAP, Reuse and Memory Access Cost aware eviction policy, that takes cache line reuse characteristics and memory access behavior into consideration when making cache line eviction decision. This allows ReMAP to mitigate the two undesirable effects described above and achieve higher performance compared to other recency based policies.

### III. Reuse and Memory Access Cost Aware Cache Replacement

Conceptually in an LRU-based cache replacement policy, each cache line in a cache set is given a reuse counter that records how long ago the particular cache line was last reused. For example in a 16-way cache, each cache line in a set is assigned a number between 0 and 15. Every time a cache line is accessed, its counter is reset to zero while all other cache lines’ counters increase by 1. When a cache line needs to be replaced, the eviction candidate is selected by choosing the cache line that has the largest counter value. In ReMAP, instead of assigning a predetermined counter value as in LRU, we assign a cost to each cache line, indicating the cost of evicting a particular cache line versus keeping it in the cache. For example, the cost is higher if a cache line to be evicted will experience a longer memory access latency when it is accessed next time. Therefore when selecting an eviction candidate, ReMAP looks at the cost for each cache line in the set and picks the line that has the least cost, contrary to an LRU-based system.

ReMAP determines the cost of the eviction candidate by considering a cache line’s recency ($R$), predicted post eviction reuse distance ($PERD$), and memory access cost ($MAC$). While the recency information gives us an insight into the line’s liveliness when the line is still in the cache, PERD provides us with additional information about how soon a line would be recalled into the cache after it has been evicted. Finally, MAC provides additional information on the associated latency for main memory access when the line gets recalled. These three vital pieces of information help in assessing the worthiness of a cache line. At the time of eviction, an effective cost of each cache line is determined using a linear relationship between the aforementioned parameters.

$$\text{Effective cost} = \alpha \ast R + \beta \ast \text{PERD} + \gamma \ast \text{MAC} \tag{3}$$

Intuitively, the cache line with the least Effective cost is less important. Therefore, ReMAP always selects the cache line with the least Effective cost for eviction. The pseudo code for cache line eviction selection is illustrated in Algorithm 1.

#### A. Recency Estimation

Recency ($R$) is typically available from the underlying cache replacement policy. For example, the recency counter in a LRU based cache indicates how recently a cache line is or will be used. Similarly, in another state-of-the-art cache replacement policy, RRPV [14], the re-reference interval predicted value (RRPV) provides a measure of the recency of a cache line. We use RRPV in estimating a cache line’s recency ($R$) component of the effective cost calculation.

#### B. Post Eviction Reuse Distance Estimation

To learn a cache line’s reuse behavior and predict the post eviction reuse distance, we use a bloom filter [4] based victim buffer that records the address of every cache line that is evicted from the cache. Upon every cache miss, the victim buffer is looked up for the missing line address. We empirically design the victim buffer to hold cache entries. In order to facilitate our PERD estimation with a practical hardware overhead, we design the victim buffer by cascading three bloom filters in a hierarchical fashion as shown in Figure 2. Upon eviction, a cache line is inserted into the first stage of the victim buffer. When the number of entries in the victim buffer is equal to one-third the number of entries in the cache, the entries in each stage is flushed down to the stage below it.

To classify the PERD for the entries in the victim buffer, we use the following heuristics. If the missing address is found within the first 1/3 of cache entries (Stage 1 BF in Figure 2) in the victim buffer, the line’s PERD is predicted to be “near”. If the missing address is found within 2/3 of cache entries (Stage 2 BF in Figure 2), the line’s PERD is predicted to be “intermediate”. If the missing address is found within cache entries (Stage 3 BF in Figure 2), the line’s PERD is predicted to be “far”. If the missing address is not found within cache entries, the line is predicted to have no reuse, or “dead”.

![Algorithm 1: ReMAP eviction decision algorithm.](image)

#### Algorithm 1: ReMAP eviction decision algorithm.

```plaintext
Input: EvictionSet
// make eviction decision
for all lines in EvictionSet do
    cache.line.EffectiveCost = \alpha \ast cache.line.R + \\
    \beta \ast cache.line.PERD + \gamma \ast cache.line.MAC.
    if cache.line.EffectiveCost < cache.MinCost then
        cache.MinCost = cache.line.EffectiveCost;
        cache.EvictionCandidate = cache.line;
end
// insert to victim buffer
insert_to_vb(cache.EvictionCandidate);
```

### Fig. 1. Memory access cost and oracle reuse information of evicted lines at LLC for SPEC2006 benchmarks.
This predicted PERD is recorded with each cache line using two additional bits. The PERD encoding is described in Table I. The hardware overhead of the victim buffer is described in the Section 4 and 6.2.3.

C. Memory Access Cost Determination

In addition to obtaining the recency and PERD information, we need to obtain the associated memory access cost. To obtain the MAC, we use a small auxiliary structure called MAC estimation table. The MAC estimation table holds a small memory access trace of current reference and previous references by storing the row addresses of two references preceding the current reference for each bank, and the number of references waiting to be serviced by each bank. At the time of insertion, based on whether the bank has requests waiting and if the current row address matches the previous two row addresses, the MAC is determined as described in Table II. The predicted MAC of the new inserted cache line is then recorded and used to calculate the overall effective cost at the time of eviction.

D. \( \alpha, \beta, \) and \( \gamma \) Parameters in \( \text{EffectiveCost} \) Computation

The parameters \( \alpha, \beta, \) and \( \gamma \) in the \( \text{EffectiveCost} \) calculation (Equation (3)) represent the importance of each of the three pieces of knowledge, i.e., recency, post eviction reuse distance, and memory access cost, for estimating the cost of LLC misses. We qualitatively explore different combinations of values for these parameters exhaustively. We evaluated ReMAP for setups that give equal importance to \( \alpha, \beta, \) and \( \gamma \), higher importance to one of the three, and lastly, giving higher importance to two of the three, pieces of knowledge. Our results show that while some applications benefit from \( \alpha = 1, \beta = 1, \gamma = 4 \) and some other applications benefit from \( \alpha = 1, \beta = 4, \gamma = 1 \). Therefore, we implement a set dueling mechanism that dynamically selects the eviction policy that minimizes the total memory access cost (as compared to number of misses in traditional set dueling schemes).

IV. IMPLEMENTATION AND HARDWARE OVERHEAD

Input: cache_access

// Upon cache miss
1. Issue request to DRAM
2. Compute Effective Cost for all cache lines (Off critical path)
3. Find eviction candidate with minimum effective cost (Off critical path)
4. Perform PERD estimation victim buffer access for missing address (Off critical path)
5. Perform MAC estimation for missing address (Off critical path)

// Upon cache insertion
1. Attach PERD value to new cache line
2. Attach MAC value to new cache line
3. Complete cache insertion

Algorithm 2: ReMAP algorithm.

Algorithm 2 shows the ReMAP algorithm and Figure 3 shows the hardware structures used in ReMAP. ReMAP uses the PERD estimation victim buffer which is a multi-level bloom filter. Each cache line has two 2-bit fields to record PERD and MAC estimations.

The PERD estimation victim buffer is implemented as a set of three bit arrays as shown in Figure 2. These bit arrays are of size \( 5 \times c \) bits, where \( c = 10 \). Each cache line inserted into the victim buffer is represented by \( k \) bits. These “k” bits are identified by a set of “k” hash functions. The victim buffer hardware overhead for the above setup is 18.75 KB. This additional hardware requirement is reasonable given the significant performance gain and is comparable to the hardware overhead of recently proposed state-of-the-art replacement policies [6][28][29].
Apart from the PERD estimation victim buffer, ReMAP consists of negligible logic overhead from the effective cost calculation and 4 bits per cache line to store the line’s MAC and PERD values.

V. Experimental Methodology

A. Simulation Infrastructure

We evaluate ReMAP using an open source full system simulator, gem5 [3]. We model a 4-way out-of-order processor with a 128-entry reorder buffer, a three-level non-inclusive cache hierarchy, and a multi-channel, multi-bank DRAM. The memory hierarchy is based on an Intel Core i7 system [10]. The L1 and L2 caches are private to each core and implement the LRU replacement policy. The configurations of our setup is summarized in Table 3. This setup is similar to the setup used in other recent studies [1], [5], [6], [12], [15], [18], [20], [24], [31].

We build ReMAP on top of a recently proposed cache replacement policy, Static Re-Reference Interval Prediction (SRRIP) [14] because SRRIP requires less hardware overhead than LRU and outperforms LRU [14].

B. Workload Construction

We evaluate ReMAP for both sequential and multiprogrammed workloads.

1) Sequential Workloads: For our study with sequential workloads, we use 8 memory sensitive (MS), and streaming or large working set (Str) benchmarks from the SPEC2006 benchmark suite. We use Simpoints [22] methodology to identify single 250 million instruction representative region for each benchmark and use this for our study. Table 4 shows the benchmarks used in our study.

2) Multiprogrammed Workloads: For our study with multiprogrammed workloads, we add one memory sensitive (MS), two streaming (str), and three compute intensive (CI) benchmarks to model realistic multiprogrammed application execution scenarios. We choose 4-core combinations of the workload types and create 18 workload mixes such that all combinations of the different types are covered. The workload mixes are shown in Table 5.

For our multiprogrammed simulations we fast-forward two billion instructions from the program start and simulate in detail until all the benchmarks have simulated for at least 250 million instructions. The benchmarks continue to run after they have finished executing 250 million instructions until all other benchmarks within that set have completed simulating 250 million instructions. This is done so that the faster benchmark continues to offer cache contention while the slower benchmark is running. However, in such a case, the statistics are collected only for the first 250 million instructions.

VI. Experimental Results

We evaluate ReMAP by comparing its performance with LRU, DRRIP [14] and the MLP-aware replacement policy (MLP-aware) [25] as these policies are most closely related to ReMAP. We perform sensitivity studies to determine the weights given to R, MAC, and PERD in the effective cost computations. We observe that the optimal configuration varies from application to application. Therefore, we use set dueling [24] to determine the weights for MAC, R, and PERD dynamically at runtime. Specifically, for the effective cost computation, we employ set dueling between “α = 1, β = 1, γ = 4” and “α = 1, β = 4, γ = 1” for our performance studies. Having values which are powers of two for β and γ makes the hardware implementation simpler. Unlike in [24], the policy selector counter is updated based on the total MAC incurred by the component policies instead of the total number of misses incurred.

A. Sequential Workloads

Figures 4 and 5 compare the cache miss reduction and IPC improvement experienced by the sequential workloads under the different policies: DRRIP, MLP-aware, ReMAP-16, ReMAP-best-VB. ReMAP-16 represents the performance of ReMAP with 16 entry per set victim buffer and ReMAP-best-VB represents the performance of ReMAP when victim buffer size is fine tuned for each benchmark. To identify the best victim buffer setup for each benchmark, we search through victim buffers having 8 through 64 entries per cache set. Though most applications are found to perform best when the victim buffer sizes are less than 24 entries, applications such as cactusADM, mcf, and soplex perform best with victim buffers containing up to 48 entries per cache set.

While all three policies reduce application LLC misses, the miss reduction does not always translate to IPC performance improvement. For example, for sphinx3, DRRIP reduces the number of misses the most, but because of the memory access cost disparity among the misses, the benefit of miss reduction is not reflected in application IPC performance improvement. Overall, ReMAP reduces the number of misses of SPEC2006 applications by as much as 13% over the baseline LRU replacement and by an
average of 6.5% while MLP-aware replacement and DRRIP reduce the miss counts by -0.7% and 5% respectively. More importantly, when looking at application IPC performance improvement, ReMAP-best-VB achieves an average of 4.6% performance gain across the SPEC2006 applications while MLP-aware replacement and DRRIP see only 1.7% and 2.3% respectively. Here onward, to maintain generality we only discuss the results of ReMAP-16.

To illustrate the importance of considering the post eviction reuse distance and memory access cost in LLC management, we take a closer look at cactusADM. Figure 4 shows that all three replacement policies reduce the LLC miss count for cactusADM by 3-5%. However, because not all cache lines are equally important, the LLC miss count reduction does not translate to IPC performance improvement linearly. Figure 5 shows that ReMAP improves the performance of cactusADM by 2% while DRRIP and MLP-aware improve its performance by 0.1% and 0.5% respectively. The reason for the IPC performance gap can be explained by Figure 1. The figure shows that, for 40% of cache line evictions, a live line with higher memory access cost is chosen as the eviction candidate under LRU. In contrast, ReMAP is able to identify and prioritize cache lines with higher memory access cost over those with lower memory access cost.

ReMAP adopts a holistic approach towards LLC management and this is highlighted in the cases of libquantum and soplex. For both these applications all three policies achieve similar MPKI reduction. However, ReMAP achieved superior IPC improvement compared to DRRIP and MLP-Aware policies.

B. ReMAP Sensitivity to Various Design Choices

1) Benefit of Using PERD and MAC Information in Isolation: In order to understand the contributions of each of the individual components of ReMAP, i.e. the post eviction reuse distance and memory access cost, we study the performance benefit achieved by each component in isolation for a few interesting applications. Figure 6 shows the performance results of using PERD and MAC information in isolation, on top of SRRIP. SRRIP-PERD represents the setup where only PERD information is used to make eviction decisions along with the recency information. Similarly SRRIP-MAC represents the setup where only MAC information is used to make eviction decisions along with recency information. Figure 6 highlights the importance of the considering all three parameters, recency, PERD, and MAC, together while making the eviction decision.

2) Sensitivity to Victim Buffer Storage: The multi-level bloom filter based victim buffer consists of three stages of bloom filters cascaded together. It can be seen that the hardware overhead depends largely on the number of bit array size of the bloom filter. As the bit array size increases, the bloom filter false positive rate decreases. We observe that we can achieve a reasonable accuracy (false positive rate 1%) when the bit array size is 10x the number of entries in the victim buffer. Furthermore, as the number of entries in the victim buffer increase, we will be able to capture the reuse behavior more accurately. However if the number of entries is very large, the reuse behavior can be misguided and in turn hurt performance. We observe a similar trend in our experiments. This behavior can be seen in Figure 7.

C. ReMAP Sensitivity to Various System Parameters

ReMAP performance can be influenced by system parameters such as the baseline replacement policy and memory scheduling policy. In addition to RRIP, we conducted experiments with ReMAP built on top of the LRU
policy. We observe that ReMAP shows similar performance improvement when the recency information is provided by LRU.

We also conducted studies to understand the sensitivity of ReMAP to architectural parameters such as cache sizes and cache set associativities. We conduct our sensitivity study with cache sizes from 1MB through 32MB, and associativities from 16-way, through 64-way configurations. We observe that ReMAP continues to provide significant performance benefit ranging from 2% to 7% on average and as high as 25% in case of benchmarks such as libquantum and mcf.

The performance of ReMAP can be sensitive to the underlying memory scheduling policy. Different memory scheduling policies can alter the MAC of the cache lines differently. Furthermore, estimating MAC under more sophisticated policies can be non-trivial. In such cases, MAC value can be communicated from the main memory to the last level cache with negligible overhead on the bandwidth demand. We expect that the MAC information and PERD information will continue to be important pieces of information that can assist LLC management even under more sophisticated memory scheduling policies as well.

D. Multiprogrammed Workloads

We evaluate the heterogeneous multiprogrammed workloads for overall system throughput and fairness. We measure overall system throughput using the normalized average throughput and normalized average weighted speedup metrics. The normalized average throughput is given by \( \frac{GM(IPC_{\text{policy}})}{GM(IPC_{\text{LRU}})} \), for \( i = 0, 1, 2, 3 \). This metric indicates the overall throughput improvement of the system. We use the minimum normalized throughput achieved across all threads as a fairness metric. This metric gives a conservative measure of fairness of the new policy relative to the fairness of the baseline. This is given by \( \min_i \left( \frac{IPC_{\text{policy}}}{IPC_{\text{LRU}}} \right) \).

Figure 8 summarizes the normalized average throughput achieved by ReMAP, MLP-Aware [25], and TA-DRRIP [13] policies for different workload mixes. Across all workload mixes, ReMAP improves average throughput by 2.5% compared to LRU and TA-DRRIP improves by 1.8%. MLP-Aware policy by -14% compared to LRU.

Overall ReMAP performs better than both TA-DRRIP and MLP-aware policies on the fairness metric as well. ReMAP gives a normalized minimum throughput of 0.9 compared to LRU while TA-DRRIP and MLP-aware policies give 0.8 and 0.7 respectively.

VII. RELATED WORK

Though we cannot summarize all the innovations in cache management research [1], [2], [5], [6], [8], [12], [15], [18], [20], [21], [26], [24], [31], we discuss prior works that closely resemble ReMAP in this section.

A. Reuse Distance Prediction

Jaleel et al. [14] proposed SRRIP and DRRIP to learn reuse behavior of applications and manage the last level cache accordingly. DRRIP provides both scan and thrashing resistance by performing set-dueling [24] between its two component policies, SRRIP and BRRIP. SRRIP provides scan resistance by inserting cache lines with “long” reference interval prediction. BRRIP on the other hand, provides both thrashing resistance by predicting “distant” re-reference interval most of the times and “long” re-reference interval infrequently. The recently proposed EAF-cache [28] predicts whether a cache line will receive reuse or not at the time of insertion, based on it’s own past behavior. Though RRIP and EAF-cache predict the reuse behavior of cache lines, their predictions are limited to insertion time. On the contrary, ReMAP uses post eviction reuse distance prediction. This helps ReMAP to predict if a cache line will be recalled to the cache or not, and also how soon would a line be recalled once it is evicted from the cache.

Rajan et al. proposed shepherd cache [27] to emulate optimal replacement in the cache. They use four shepherd ways in a 16-way cache that will keep track of partial reuse distances and allowing to evict the the lines that are reused farther into the future. Their proposal allows for limited look ahead and ReMAP on the other hand is able to predict reuse distances much which are much farther, up to three times the associativity of the cache.

B. Dead Block Prediction

Many works have also used a variation of recency, instruction traces, or address traces to predict blocks that are dead [9], [16], [19]. Sampling Dead Block Prediction [17] proposed by Khan et al. predicts cache blocks that are dead in the cache based on the last touched instructions. They replace these predicted dead blocks prior to the LRU replacement candidate. Chaudhuri et al. proposed cache hierarchy-aware replacement (CHAR) [6] policy where they mined the private L2 cache eviction stream for information that identifies certain blocks to be dead and passed the hint to the LLC.

While ReMAP’s PERD estimation is similar to dead block prediction, it is important to note that ReMAP predicts the reuse distance in a finer granularity, and this paper shows that the finer-grained PERD prediction contributes to further performance improvement.

C. Coordinating LLC Management with DRAM

Qureshi et al. first identified the potential in considering DRAM access costs in managing the LLC in [25]. They assign cost to each cache line based on the amount of memory level parallelism the cache line would present if it misses in the cache. They adopt a linear relation-ship between recency and the MLP cost of cache lines to determine the total cost. We present a more fine-grained memory access cost analysis and combine that with post eviction reuse distance along with recency information to assign effective costs to cache lines. This coordinated approach enables ReMAP to provide higher performance improvement than MLP-aware replacement.
VIII. Conclusion

The miss rate reduction achieved by most state-of-the-art cache management policies does not translate to corresponding IPC improvement at all times. This is because of the wide disparity in memory access costs experienced by different LLC misses. Hence, it is vital to manage the last level cache while considering memory access behavior of cache lines. Furthermore, the system performance is understandably affected by the current and future reuse of cache lines. With this insight, we proposed ReMAP, a reuse and memory access cost aware eviction policy that takes recency, post eviction reuse distance, and memory access costs to make better-informed eviction decisions at LLC. ReMAP is able to preserve most valuable cache lines, i.e., lines that have near reuse and high memory access cost, in the LLC and thereby providing superior performance. We demonstrate with extensive performance evaluation using wide variety of workloads that ReMAP performs consistently better than related state-of-the-art policies such as MLP-aware replacement, DRRIP and TA-DRRIP.

Acknowledgment

We would like to thank the anonymous reviewers for their feedback. This work is supported by Arizona State University.

References