IPKB: A New Metric for Evaluating Cache Replacement Policies

Alireza Haddadi
department of computer architecture
university of Isfahan
Isfahan, Iran
a71.haddadi@gmail.com

Mehran Rezaei
department of computer architecture
university of Isfahan
Isfahan, Iran
m.rezaei@eng.ui.ac.ir

Hooman Nikmehr
department of computer architecture
university of Isfahan
Isfahan, Iran
nikmehr@eng.ui.ac.ir

***Abstract*—Nowadays replacement policies draw the attention of researchers. These policies have a direct impact on the cache miss rate and consequently on the performance and power consumption. The hardware overhead grows as the complexity of cache replacement policies increase which results in energy penalties. In this article, we introduce IPKB (Improvement Per Kilo Byte) as a touchstone for replacement policies and then use this new metric to evaluate recent common and credible replacement policies. IPKB indicates both miss rate and hardware overhead, the latter is a better representative of energy consumption. We show that using our metric, the policies with higher miss rate improvement were not the best policy because of their massive hardware overheads.**

Keywords—replacement policy, cache, memory systems, memory power consumption, replacement policies' metrics

I. Introduction

Researchers and scientists no longer have concerns about the computation speed because of the advent of multicore processors and recent advances in this domain. Today, the main concern is about the well-known memory-wall problem. The memory response time is much slower than the processor. Computer pioneers addressed this problem by using memory hierarchy and the cache.

Cache sits between the main memory and the processor and provides data for the processor faster than the main memory. Since the capacity of the cache is much smaller than the main memory, the data that processor no longer needs must be replaced by the data which is needed. Replacement policies choose which block of cache should be replaced. For example, common and well-known LRU replacement policy chooses the block that is not recently used by the processor. Researchers designed several replacement policies to improve LRU or to address LRU's problems and reduce the miss rate of the last level cache (LLC). Every replacement policy imposes a hardware overhead to the cache and consequently adds to the power consumption of the memory system. This hardware overhead limits the complexity of the policies.

Researchers who develop their own policy, evaluate and examine their policies by common metrics like MPKI (Miss per Kilo Instruction) or IPC (Instruction Per Cycle) to compare with other policies. These metrics can show how efficient a replacement policy is, however, there is another flip side to this coin. The added hardware overhead which plays a vital role in power consumption and area is considered by neither MPKI nor IPC. We Introduce IPKB as a multilateral metric for replacement policies which considers both miss rate improvement and hardware overhead. We evaluated five common and recent replacement policies by this new metric which are LRU, SRRIP, DRRIP, SHiP, and Hawkeye.

II. Literature Review

Considering recent credible researches on cache replacement policies [1-7] and its metrics, to our knowledge, this is the first study on metrics whose evaluations so far evaluate the efficiency of replacement policies. There have been considerable innovations and researches on the development of replacement policies, such as the "Harmony" [8] approach and "Global History Reuse Prediction" (GHRP) [9] which improve miss-rate in the presence of memory prefetchers. In this segment, we review five of the most credible and relevant recent works which their evaluations are totally based on miss rate improvement.

A. Least Recently Used (LRU)

LRU and its derivations [10-12] are the gold standards in this field of research. The goal is to choose a victim block for replacement based on the recent usage of the blocks. The blocks are sorted in a sequence from 0 to N-1 (i.e., LRU chain, depicted in Fig. 1), where N is the cache associativity. When the processor uses a block, it is moved to the head of chain (numbered 0) and the blocks resident prior to the previous position of the used block will be pushed one place further down in the LRU chain. When there is a need for replacement, the block at the tail of LRU chain is the victim and will be replaced. This policy performs poorly in terms of thrashing and mixed access patterns.

Mixed access patterns consist of a working set with a following non-temporal sequence that is accessed by the CPU only once and evicts the working set part from the cache without any participation in hits.

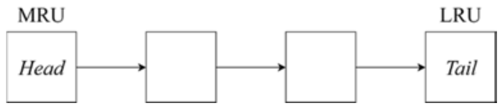


Fig. 1. LRU chain of a 4-way set associative cache

To better understand how poorly LRU performs with thrashing access patterns, let [a,b,c,…] denote a temporal sequence of references, $[a,b,c,\ldots,e]^N$ denote a temporal sequence of references that repeats N times, and $[a,b,c,d,e]^N$

978-1-7281-1508-5/19/$31.00 ©2019 IEEE

denote the time when block c is accessed. Fig. 2 presents the LRU-chain of a 4-way set associative cache.

B. Dynamic Insertion Policy (DIP)

DIP [13] can protect the cache when access patterns are thrashing and reduces miss rate. This policy uses two components namely BIP and LRU. Upon an insertion, BIP component mostly places the incoming line in the tail of the LRU chain and infrequently (with low probability) places the incoming line in the MRU position. In this case, when the working set is larger than the cache size, a big part of the working set is preserved by cache and therefore the cache is protected from thrashing access patterns.

Based on the locality behavior of applications, we can choose either BIP or LRU, since some applications perform better under BIP implementation of replacement policy while others perform better under LRU. Using Set Dueling, DIP dynamically chooses between BIP and LRU, depending on which policy incurs fewer misses.

Set Dueling method categorizes the cache sets into three groups. The first group uses LRU policy upon an insertion, i.e., the incoming block is placed to the MRU position. The second group uses BIP policy, where the third one, which is called follower sets, uses the policy (BIP or LRU) that incurs fewer misses.

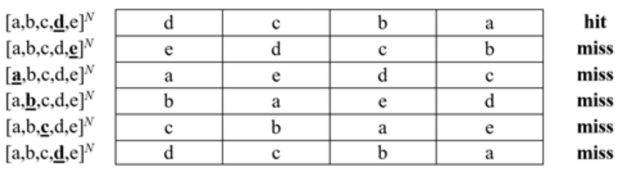
DIP uses an n-bit counter to show which policy produces fewer misses. All of the bits of the counter are set to 1 except the most significant bit which is kept 0. A miss in the first group of sets decrements the counter while a miss in the second group increments it. Finally, the most significant bit (MSB) of the counter indicates the winner policy. MSB = 1 denotes that BIP results in more misses, thus the follower sets must use LRU. Fig. 3 depicts the architecture of the DIP policy.

Although DIP protects the cache against thrashing access patterns, this policy still performs poorly when the applications' data access patterns contain multiple scans.

C. Re-Reference Interval Prediction (RRIP)

As it is obvious, an optimal replacement policy would preserve the working sets whenever scans occur and would not let the scans evict the working sets from the cache. Therefore after the scans, any reference to the recent working set has a chance for participating in hits. RRIP policy [14] tries to make the cache scan-resistant.

In this policy, one m-bit value is assigned to each cache block to store one of 2^m possible Re-Reference Prediction Value (RRPV). It predicts that blocks with small RRPVs are referenced sooner than the blocks with large RRPVs. RRIP policy has two derivations: Static RRIP and Dynamic RRIP.



$[a,b,c,\underline{d},e]^N$	d	c	b	a	**hit**
$[a,b,c,d,\underline{e}]^N$	e	d	c	b	**miss**
$[\underline{a},b,c,d,e]^N$	a	e	d	c	**miss**
$[a,\underline{b},c,d,e]^N$	b	a	e	d	**miss**
$[a,b,\underline{c},d,e]^N$	c	b	a	e	**miss**
$[a,b,c,\underline{d},e]^N$	d	c	b	a	**miss**

Fig. 2. LRU chains when there is a thrashing access pattern

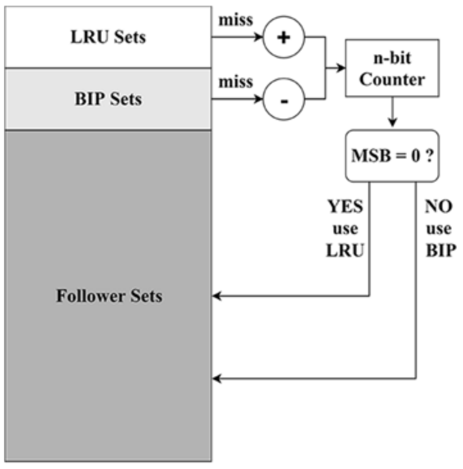


Fig. 3. DIP Replacement Policy

In SRRIP policy, upon every insertion of a block, the value 2^m-2 is assigned to the block's corresponding RRPV. This means that SRRIP predicts that all blocks have long distant re-reference intervals. Upon a cache hit, the value of the RRPV corresponding to the hit block is set to 0. When there is a need for block replacement, SRRIP searches for a block with corresponding RRPV = 2^m-1. If there is no such value, SRRIP increments all RRPVs and repeats the search until it can find a block with RRPV = 2^m-1. Although this policy is scan-resistant, it cannot handle thrashing access patterns.

DRRIP is thrash-resistant and an enhancement to SRRIP. DRRIP consists of two components, BRRIP (Bimodal RRIP) and SRRIP, which obviously behaves as explained above. BRRIP mostly (not always) sets the RRPV to 2^m-2 and infrequently set the RRPV to 2^m-1 upon an insertion. Like DIP, there is a Set Dueling between SRRIP and BRRIP, and the winner method is chosen as the replacement policy of the follower sets. Fig. 4 is an illustration of DRRIP policy architecture.

All cache blocks upon an insertion, have a static value of 2^m-2 or 2^m-1 regardless of their behavior in the past. Therefore the prediction is almost the same for all blocks before or after they moved to the cache (RRIP tries to correct its prediction if needed). In other words, the learning phase is accomplished only after the insertion.

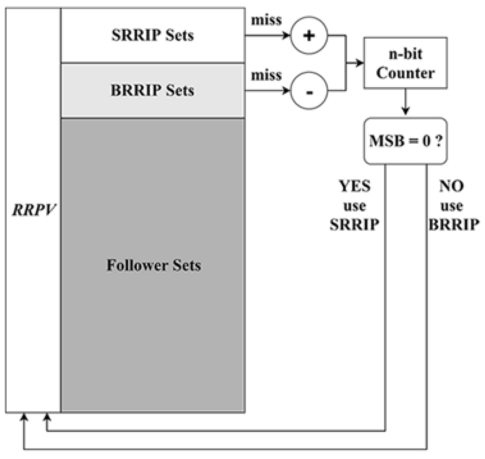


Fig. 4. RRIP Replacement Policy

D. *Signature based Hit Predictor (SHiP)*

SHiP [15] improves RRIP by adding a table of counters and two more fields to RRPV, Signature, and Outcome-bit. Signature is used to somehow store PCs responsible for memory references even if the referenced data is not currently in the cache. Outcome-bit is a one-bit value which indicates that a block participates in cache hits or not. In other words, outcome bit = 1 indicates that the corresponding block caused at least 1 hit, since insertion and in the same way the value of 0 indicates that the corresponding block is never reused since its insertion. The table of counters indexed by the signatures and its value indicates the behavior of the blocks brought into the cache. These blocks correspond to PC values of the load-store instructions responsible for their insertion.

Upon a block insertion, encoded PC is assigned to the signature field and the outcome-bit is set to 0. Upon a cache hit, the corresponding counter is incremented and the outcome-bit is set to one which means that this block is used at least once after its insertion. Upon a block eviction, the corresponding counter is decremented if the outcome-bit is zero which means that the only possible way that a counter is decremented is that the corresponding block is never reused between the time of insertion and eviction.

At the time of a block insertion, its RRPV is set to 2^m-1 only if the corresponding counter has value of zero; otherwise, the RRPV will be set to 2^m-2. Obviously, in SHiP the prediction of the RRPV is not haphazard rather it is based on the previous behavior of the block.

Because of the limitation of hardware budget, there is no chance to assign signatures to all of the cache lines and to keep the large table of counters beside the cache. Thus only 64 lines of the cache are given signatures which are called Sampling sets. Finally, there is a Set Dueling between LRU and SHiP policies and the winner is picked for the follower sets. Fig. 5 illustrates the SHiP policy.

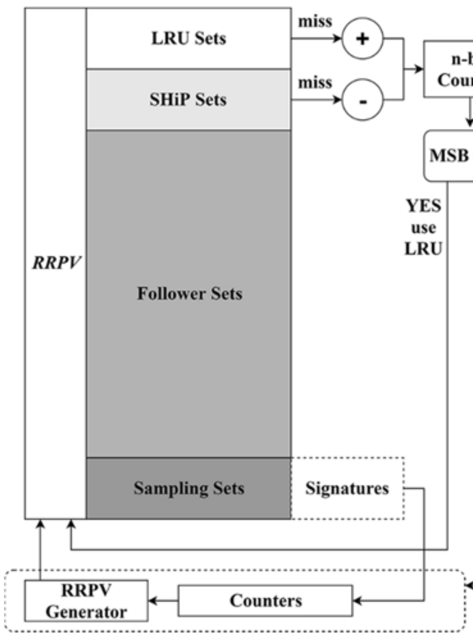


Fig. 5. SHiP Replacement Policy

E. *Hawkeye*

This policy [16] has a method to decide about the value of RRPV upon a block insertion (same as SHiP) which is inspired by Belady's algorithm [17]. There is an OPTgen unit beside the cache in this policy that categorizes the cache blocks to cache-friendly and cache-averse. The cache-averse blocks, upon insertion, have RRPV value of 2^m-1 (maximum value) and the cache-friendly blocks have the RRPV value of zero (minimum value).

Belady's algorithm, the optimal replacement policy, needs future information about references to decide which cache block should be evicted. Since capturing the future information is not possible, Belady's algorithm is not practical; however, the OPTgen unit in the Hawkeye policy uses this algorithm for the past memory references to determine which one would have been cached if the Belady's algorithm had been used. Thus, a block is cache friendly if it would have been cached by Belady's policy and is cache averse otherwise. Hawkeye policy shows that more than 90% of the memory references will have the same behavior in the future.

To fully implement OPTgen, some space which is 8 times larger than the cache tag area is needed. The needed space is used to store the past references. Due to hardware budget limitations, it is impossible to implement full OPTgen; therefore, only 64 lines are sampled to model the entire cache.

Similar to the previous replacement technics, there is a set dueling between Hawkeye and LRU and the winner policy is used for the follower sets as depicted in Fig. 6.

III. IPKB: A New Metric For Evaluating Cache Replacement Policies

Nowadays, according to recent studies on multi-processor chips, caches consume the maximum amount of total power consumed by the chip. Thus caches play the main role in the chip power consumption and consequently its temperature which influences the performance of the chip directly.

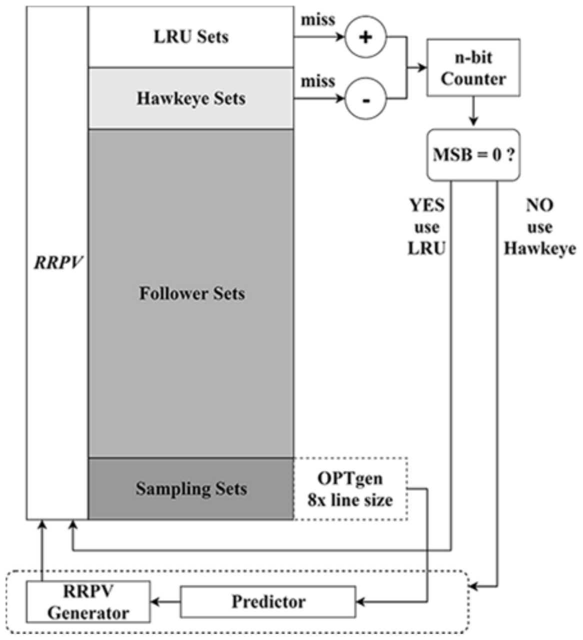


Fig. 6. Hawkeye Replacement Policy

Replacement policies have a direct impact on cache power consumption and its area. Studies show that, as the complexity of the replacement policies increased, more hardware overhead is needed and therefore more power is dissipated by the cache.

Today, many of scientists and researchers develop their own replacement policy and compare them with the others by using common metrics such as miss-rate, miss-rate improvement over LRU, miss per kilo instructions (MPKI), and instruction per cycle (IPC). Although these metrics can evaluate and examine replacement policies' performance, lack of dependency on hardware overhead (hardware added to the main structure of the cache to implement replacement policy) in these metrics makes this evaluation inaccurate. Since there are concerns about the chip's power consumption, a metric like miss-rate cannot satisfy researchers in all aspects needed by them. This issue has motivated us to craft a metric that can differentiate replacement policies with different hardware overheads.

On the first attempt, we determined "Improvement over LRU Per Kilo Byte" (IPKB) could be a multilateral metric for replacement policies. This metric computes the miss rate improvement over LRU per each Kilo Byte of the hardware added to the structure of the cache. While MPKI, which was used for evaluating all mentioned replacement policies, only focuses on miss rate improvement, our new metric considers hardware overhead an important factor, as well as miss rate improvement. (1) shows the IPKB equation.

$$IPKB = \frac{miss\ rate\ improvement\ over\ LRU}{total\ hardware - LRU\ hardware} \qquad (1)$$

After some experiments, we recognized that the new metric is not applicable to replacement policies with lower hardware overhead than the common LRU. Therefore we modified our evaluation method and named it weighted IPKB which is applicable for replacement policies with both larger and smaller hardware overheads. (To shorten our equations, instead of MPKI improvement over LRU, we write m_improvement in the next equations). (2) shows the weighted IPKB equation.

$$IPKB = \frac{LRU\ overhead}{new\ hardware\ overhead} \times m\_improvement \quad (2)$$

Considering our new approach, higher miss rate improvement and lower hardware overhead, which are both desirable, result in higher IPKB. Therefore to our calculations, higher IPKB specifies the better policy. The policy with better improvement in miss rate is not always the winner policy because it might impose great hardware overhead to the cache structure.

Hardware overhead can be calculated by adding up the space required for keeping the policy's meta-data. As an illustration, assume a 2MB 16-way last level cache with 64-Byte block size which SRRIP is implemented as its replacement policy. As it is obvious, there are 32K blocks in this cache. Since RRPVs are 3-bit registers and one RRPV should be assigned to each block, therefore we need 32K×3bit=96Kb or 12KB space to implement SRRIP architecture.

If a replacement policy requires the same hardware overhead space as LRU, the only factor that influences IPKB is its miss rate improvement over LRU; however, when a replacement policy needs less hardware overhead than LRU, which is desirable, IPKB must present a higher value. These conditions for both situations are provided by (2). In our method hardware overhead normalizes the miss rate improvement. For example, if a policy with a hardware overhead of two times larger than LRU, improves miss rate by 10 percent, the miss rate improvement's influence is divided by two because of the extra hardware overhead and the fraction in (2) does this.

Up until now, to our knowledge, no researcher had looked at the hardware overhead of replacement policies. Extra hardware means more complexity, which in turn results in a slower machine ("simpler is faster") and more importantly higher energy consumption. The policy with better improvement in miss rate is not always the winner policy because it might impose great hardware overhead to the cache structure. From our point of view, the introduced metric, weighted IPKB, is a better choice for evaluating and comparing replacement policies with different hardware overhead.

In next segment, we have evaluated the four most acceptable and credible replacement policies using IPKB and compared them.

IV. Methodology

We used the ChampSim [18] simulator which models a 3-level cache hierarchy and simulates out of order CPUs. More detail about simulation conditions is given in Table I.

We calculate the hardware overhead for each replacement policy using their C++ source codes before the beginning of the simulation. Hardware overheads for each replacement policy are listed in Table II.

We used four benchmark traces and evaluated the miss rates using this simulator.

TABLE I. Simulation Setup Configuration

	Size and type	**Replacement policy**
L1 cache (I/D)	32KB, 8-way	LRU
L2 cache	256KB, 8-way	LRU
L3 cache	2MB, 16-way	Evaluating policies
Block size	64 Bytes	-
L1 prefetcher	No prefetcher	-
L2 prefetcher	No prefetcher	-
Branch predictor	Bimodal	-
DRAM	4GB	-
Core number	Single core	-

TABLE II. Replacement Policies' Hardware Overhead

Policy	**Hardware overhead**
LRU	16KB
SRRIP	12KB
DRRIP	13.6KB
SHiP	35KB
Hawkeye	31.8KB

V. Results

Fig. 7 shows the MPKI of each examined benchmark using five evaluated replacement policies. The MPKIs are large values because references to LLC are filtered by level 1 and level 2 caches and therefore the accesses exhibit no locality. We also have evaluated the MPKI improvements over LRU replacement policy and illustrated in Fig. 8. The vertical axis in this plot presents the difference between LRU MPKI and evaluating policy MPKI (LRU MPKI - other policy MPKI).The greater amount of the vertical axis means greater improvement over LRU. We finally have calculated the IPKBs for each replacement policy and the benchmarks and gathered the results in Fig. 9.

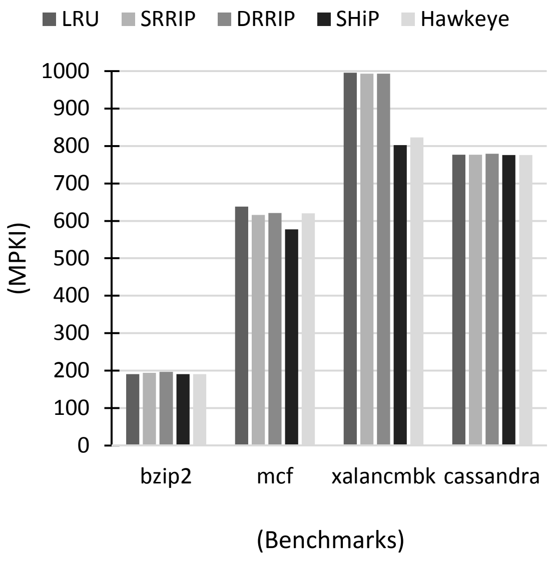


Fig. 7. Replacement Policies' MPKI

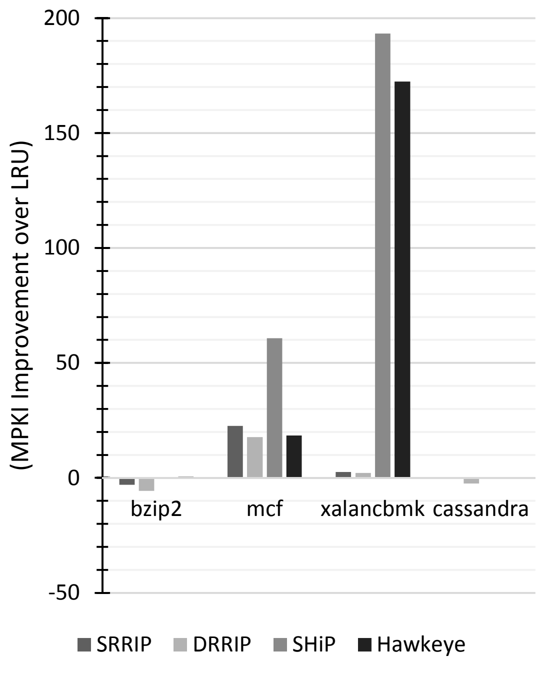


Fig. 8. MPKI Improvements Over LRU

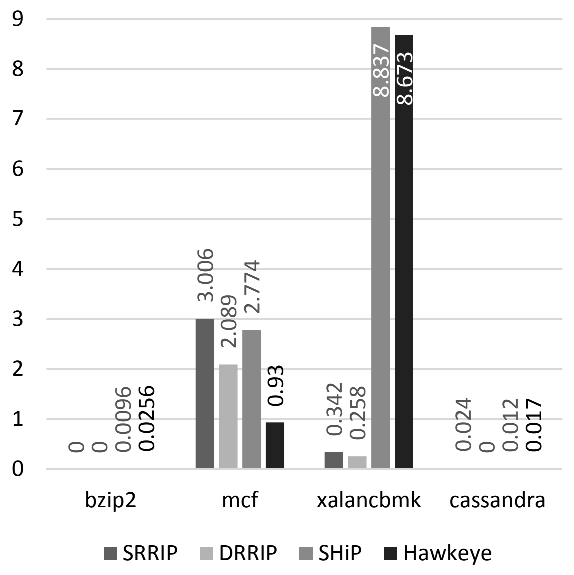


Fig. 9. Replacement Policies' IPKB

As illustrated by Fig. 8, ShiP and Hawkeye outperform LRU for "mcf" and "Xalancbmk". It is known that these two programs are graph traversal based and from the object-oriented paradigm. Therefore, in terms of temporal locality, they both don't have good behavior. For such a paradigm more aggressive replacement policy is desirable. However, if we switch our attention to Fig. 9, whereas the hardware overhead is also taken to evaluation, for even "mcf", SRRIP performs the best.

VI. Conclusion and Future Work

There is a huge amount of researches on the development of replacement policies in this domain. All of the researchers compared their methods with other policies using common metrics mostly MPKI. Although miss rate shows how efficient a replacement policy is, using this metric merely, we cannot compare two different replacement policies with different hardware overheads. In our approach, we introduced IPKB which can compare replacement policies with different hardware overheads and decide which one is the most efficient. In other words, hardware overhead can have a direct impact on the metric we use to present a policy efficiency, just the same as miss rate improvement.

In our examination, we have presented how hardware overhead can discard the MPKI improvement's effect on our new metric. We showed that using our metric, a policy with a small MPKI improvement but a large hardware overhead is not effective and desirable. Moreover, when there were no significant MPKI improvement differences between the policies, the policy with small hardware overhead had better IPKB. As a result, we showed that our new metric could evaluate cache replacement policies with higher accuracy by considering both hardware overhead and MPKI or miss rate improvement.

Note that it is not our intention to conclude which replacement policy performs the best. However, considering energy consumption as well as the design principle "simpler is faster", we set to propose that IPKB is a concrete representative metric for evaluating replacement policies which results in fair judgment and comparison. Such a metric should be able to reveal not only the performance (in this case cache miss reduction) but also the hardware overhead of policies.

As future work, it is recommended that researchers try to evaluate the energy consumption of each policy and combine this energy consumption with the IPKB and produce a new metric that can cover the energy aspect of a replacement policy just the same as miss rate and hardware overhead.

References

[1] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in Proceedings of the 42nd annual Southeast regional conference, 2004, pp. 267-272.

[2] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2012, pp. 389-400.

[3] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," Performance Evaluation, vol. 72, pp. 16-36, 2014.

[4] G. Keramidas, P. Petoumenos, and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in 25th International Conference on Computer Design, 2007, pp. 245-250.

[5] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way cache: demand-based associativity via global replacement," in 32nd International Symposium on Computer Architecture (ISCA), 2005, pp. 544-555.

[6] J. Reineke, D. Grund, C. Berg, and R. Wilhelm, "Timing predictability of cache replacement policies," Real-Time Systems, vol. 37, no. 2, pp. 99-122, 2007.

[7] J. T. Robinson and M. V. Devarakonda, Data cache management using frequency-based replacement (no. 1). ACM, 1990.

[8] A. Jain and C. Lin, "Rethinking belady's algorithm to accommodate prefetching," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 110-123.

[9] S. M. Ajorpaz, E. Garza, S. Jindal, and D. A. Jiménez, "Exploring predictive replacement policies for instruction cache and branch target buffer," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 519-532.

[10] D. A. Jiménez, "Insertion and promotion for tree-based PseudoLRU last-level caches," in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2013, pp. 284-296.

[11] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: simple and effective adaptive page replacement," ACM SIGMETRICS Performance Evaluation Review, vol. 27, no. 1, pp. 122-133, 1999.

[12] W. A. Wong and J.-L. Baer, "Modified LRU policies for improving second-level cache behavior," in Proceedings Sixth International Symposium on High-Performance Computer Architecture (HPCA-6), 2000, pp. 49-60.

[13] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," ACM SIGARCH Computer Architecture News, vol. 35, no. 2, pp. 381-391, 2007.

[14] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," ACM SIGARCH Computer Architecture News, vol. 38, no. 3, pp. 60-71, 2010.

[15] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "SHiP: Signature-based hit predictor for high performance caching," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011, pp. 430-441.

[16] A. Jain and C. Lin, "Back to the future: leveraging Belady's algorithm for improved cache replacement," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 78-89.

[17] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," IBM Systems journal, vol. 5, no. 2, pp. 78-101, 1966.

[18] “The ChampSim simulator,” [online] Available at: https://github.com/ChampSim/ChampSim.