Replacement Policy Adaptable Miss Curve Estimation for Efficient Cache Partitioning

Byunghoon Lee, Kwangsu Kim, and Eui-Young Chung, *Member, IEEE*

***Abstract*—Cache replacement policies and cache partitioning are well-known cache management techniques which aim to eliminate inter- and intra-application contention caused by co-running applications, respectively. Since replacement policies can change applications' behavior on a shared last-level cache, they have a massive impact on cache partitioning. Furthermore, cache partitioning determines the capacity allocated to each application affecting incorporated replacement policy. However, their interoperability has not been thoroughly explored. Since existing cache partitioning methods are tailored to specific replacement policies to reduce overheads for characterization of applications' behavior, they may lead to suboptimal partitioning results when incorporated with the up-to-date replacement policies. In cache partitioning, miss curve estimation is a key component to relax this restriction which can reflect the dependency between a replacement policy and cache partitioning on partitioning decision. To tackle this issue, we propose a replacement policy adaptable miss curve estimation (RME) which estimates dynamic workload patterns according to any arbitrary replacement policy and to given applications with low overhead. In addition, RME considers asymmetry of miss latency by miss type, thus the impact of miss curve on cache partitioning can be reflected more accurately. The experimental results support the efficiency of RME and show that RME-based cache partitioning cooperated with high-performance replacement policies can minimize both inter- and intra-application interference successfully.**

***Index Terms*—Cache management, cache partitioning, cache replacement policy, shared last-level cache (SLLC).**

I. Introduction

IN MODERN chip multiprocessors, shared last-level caches (SLLCs) have become a critical memory component due to their performance impact. In these systems, multiple applications with different memory requirements are executed simultaneously and the co-running applications inevitably cause interapplication contention to SLLC while each application cause intra-application contention. To solve SLLC performance degradation caused by these contentions, replacement policies, and cache partitioning are considered to be the key to minimize intra- and interapplication contention, respectively.

The research of replacement policies originally aimed to reduce intra-application interference on single-core systems, and many high-performance replacement policies have been proposed to improve cache performance of a single application over least recently used (LRU) policy which is a widely used conventional replacement policy. Such replacement policies have been extended to SLLC on the basis of per-application management [1]–[3]. However, replacement policies cannot eliminate interapplication interference on SLLC fundamentally [4].

To mitigate the interapplication interference on SLLC, cache partitioning techniques have been proposed [4]–[9]. Cache partitioning methods split and distribute the whole SLLC space into each application and guarantee them to fully occupy the allocated area, thereby eliminating interapplication interference intentionally and alleviating capacity imbalance among applications. For effective cache partitioning, an important aspect is to accurately predict the performance of various applications on SLLC with respect to allocated partition size, the change of which can be represented as *miss curve*, a discrete graph of cache miss count versus cache size. Therefore, miss curve estimation is one of the most important components for effective cache partitioning.

However, miss curve estimation is a challenging task due to its huge overhead. To obtain a complete miss curve for each application, each design point in the miss curve must be estimated, resulting in hardware or software overhead which is proportional to both the number of available partition size and the number of applications. For this reason, existing cache partitioning methods try to reduce the overhead by exploiting intrinsic operation property of a given conventional replacement policy [5], [6], so miss curve estimators of the existing cache partitioning methods are dedicated to only particular replacement policies although the choice of replacement policy is another predominant factor which affects the shape of miss curve [10]. Thus, existing cache partitioning methods may lead to suboptimal capacity allocation when incorporated with up-to-date high performance replacement policies despite opportunities to further reduce intra-application interference.

Another important observation is that misses caused by different request commands have a different performance impact on the entire system. In noninclusive SLLCs, both read misses and write misses incur memory stall time on the critical path

Manuscript received January 17, 2017; revised March 29, 2017; accepted May 10, 2017. Date of publication June 6, 2017; date of current version January 19, 2018. This work was supported in part by the National Research Foundation of Korea under Grant 2016R1A2B4011799, and in part by the IT Research and Development Program of MOTIE/KEIT (Design technology development of ultralow voltage operating circuit and IP for smart sensor SoC) under Grant 10052716. This paper was recommended by Associate Editor T. Mitra. *(Corresponding author: Eui-Young Chung.)*

The authors are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, South Korea (e-mail: eychung@yonsei.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2712666

and thus have a large impact on system performance. On the other hand, writeback misses have a negligible impact on system performance, which is similar to SLLC hits in most cases because the latency of writeback misses can be hidden. Despite these aspects, existing miss curve estimation methods do not consider this asymmetry among SLLC miss types.

To tackle the limitations, we propose replacement policy adaptable miss curve estimation (RME) which is an efficient and practical miss curve estimation method for cache partitioning which is adaptable to any arbitrary replacement policy. In RME, linear interpolation is used with a fixed number of sampling points in miss curve for low-overhead estimation. The key idea here is to change the location of the sampling point dynamically. This not only helps estimated miss curves from linear interpolation fit better to the exact miss curves but also helps the proposed miss curve estimation cope with a change of miss curve according to the dynamic characteristic of given workloads. Our contributions are threefold.

1) We propose an efficient miss curve estimation technique called RME with low overhead using linear interpolation. RME can track the change of miss curve adaptively regardless of access pattern and replacement policy.
2) RME generates *writeback-aware miss curve* which contains only the misses causing the critical path of memory stall time. Using writeback-aware miss curve, the performance impact of partition allocation can be reflected more accurately during cache partitioning.
3) RME-based cache partitioning (RCP) gives an opportunity to explore the synergy between replacement policies and cache partitioning, thus SLLC performance can be further improved by minimizing both inter- and intra-application interferences altogether.

The ultimate goal of the proposed method is to maximize the SLLC performance by amplifying the synergy between cache partitioning and replacement policy. To the best of our knowledge, this is the first work to analyze the synergy of replacement policy and cache partitioning thoroughly.

II. Related Works

A. Replacement Policy

Many replacement policies have been proposed to improve cache performance by reducing intra-application interference [2], [3], [11]–[13]. In addition, replacement policies are extended to support SLLC [1], [2]. They consider workload characteristics of individual applications and dynamically select a proper policy suitable to each application to reduce intra-application interferences of each application. However, they cannot handle interapplication interference of SLLC fundamentally, so cache partitioning methods have been proposed to solve the interapplication problem.

B. Cache Partitioning

Many cache partitioning methods have been proposed, and most of them consist of two main components: 1) an allocation policy and 2) a partitioning scheme [4]. An allocation policy logically allocates the cache capacity to each application while a partitioning scheme does the jobs of enforcing the allocated partitions. Here, miss curve estimation is a key component of an efficient allocation policy.

1) Allocation Policy With Miss Curve Estimation: Utility-based cache partitioning (UCP) [5] is one of the most well-known cache partitioning techniques which divides SLLC capacity based on application's utility of SLLC resource, represented as miss curve. UCP assumes that the replacement policy used in target SLLC is LRU and exploits LRU's stack property [14] to estimate the miss curve of each application with low-overhead. Therefore, miss curve estimator called utility monitor (UMON) generates a miss curve for each application. After collecting miss curve information, UCP finds the optimal partition allocation by mixing the estimated miss curves. Since finding optimal partition allocation is NP-hard, the authors also propose an effective heuristic allocation algorithm called *lookahead algorithm*.

Co-optimizing locality and utility (CLU) [6] extends UCP to exploit two replacement policies: 1) LRU for high locality workloads and 2) bimodal insertion policy (BIP) for low locality workloads [11]. CLU allocates SLLC capacity to applications and also dynamically selects a proper replacement policy between LRU and BIP for each application. For the allocation policy in CLU, UMON is used to estimate miss curve[1] of LRU and miss curve estimator for BIP, called locality monitor (LMON), is newly proposed since BIP do not satisfy the stack property of LRU.

UMON [5] and LMON [6] are hardware-based miss curve estimators with low overhead, and they show high accuracy on miss curve estimation for SLLC managed by target replacement policies: LRU and BIP. However, they are not able to accurately estimate miss curves of SLLC managed by other replacement policies. Furthermore, they do not classify misses according to miss type on their miss curve estimation.

2) Partitioning Scheme: An allocation policy is highly influenced by the choice of replacement policies, whereas a partitioning scheme is often easily applicable to a variety of replacement policies [4]. Way-based partitioning is a popular partitioning scheme owing to its simplicity [5], [6], [9], [15], and a partitioning scheme called Vantage [4] is proposed to overcome the poor scalability of other partitioning schemes. These partitioning schemes are compatible with arbitrary replacement policy as they can manage each partition using any given replacement policy on a per-partition basis.

In short, existing cache partitioning methods are bound to particular replacement policies due to their allocation policies, and the allocation policy must be redesigned in order to adopt other replacement policies.

C. Relationship Between Replacement Policy and Cache Partitioning

A cache performance model for arbitrary age-based replacement policy was proposed [10]. The model can accurately estimate the performance of up-to-date replacement policies with a given application and cache configuration. The authors gives a good inspiration about the relationship between replacement

[1]The authors originally uses hit curve instead of miss curve in their paper. For consistency of this paper, we substitute miss curve for the hit curve.

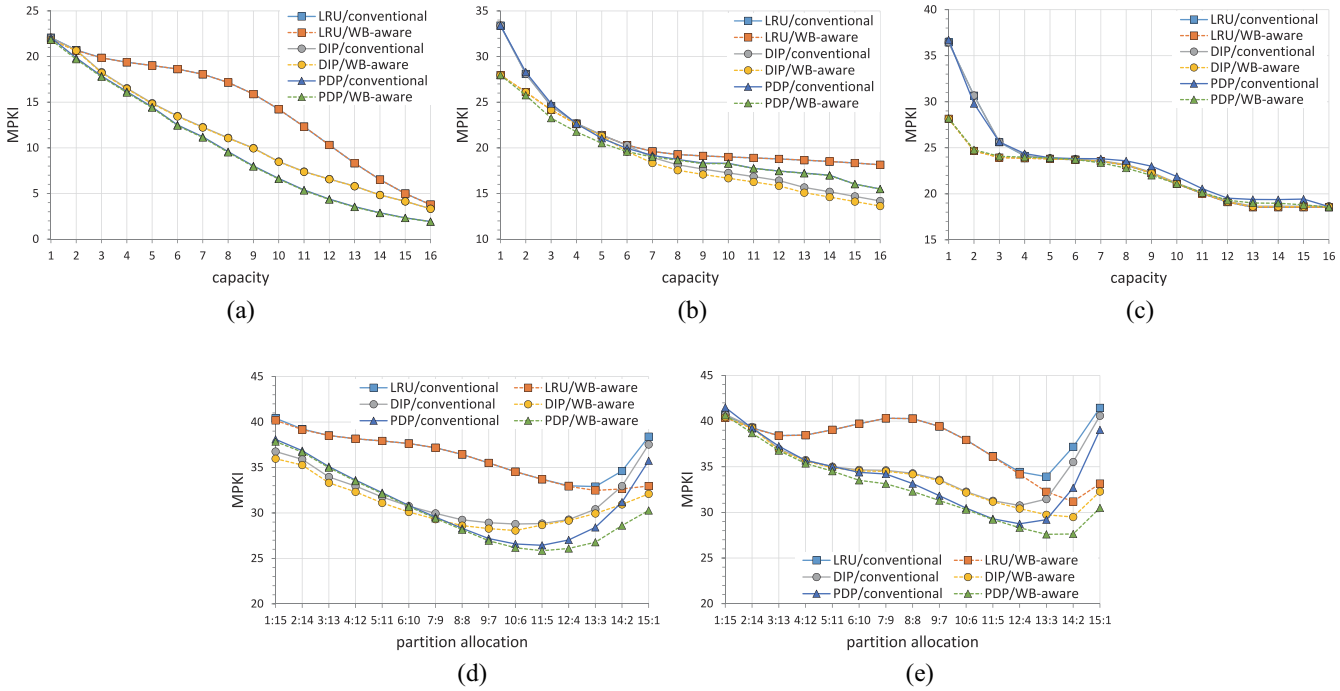


Fig. 1. Miss curves on SLLC according to application, replacement policy, and miss type. (a) Conventional and writeback-aware miss curves of *xalancbmk*. (b) Conventional and writeback-aware miss curves of *soplex*. (c) Conventional and writeback-aware miss curves of *GemsFDTD*. (d) Composite miss curves of *xalancbmk* and *soplex*. (e) Composite miss curves of *xalancbmk* and *GemsFDTD*.

policies and cache partitioning, but the allocation policy based on the model is not practical due to its scalability issue.[2]

III. Motivation

Before discussing the details of our proposed scheme, we start from motivating examples to show how replacement policies and miss types affect miss curve and how miss curve estimation affects cache partitioning.

Fig. 1 shows miss curves of three applications (*xalancbmk*, *GemsFDTD*, and *soplex*) from SPEC CPU2006 benchmarks and composite miss curves where miss curves of two applications are mixed. For each application and combination of applications, we consider two miss curves by miss types.

1) Conventional miss curve (solid line) which contains total misses by all miss types.
2) *Writeback-aware miss curve* (dashed line) which filters out misses by writeback requests which are not on the critical path of memory stall time in most cases.

Also, we consider three replacement policies: 1) LRU; 2) dynamic insertion policy (DIP) [11]; and 3) protecting distance-based policy (PDP) [3].

Fig. 1(a)–(c) show miss curves for *xalancbmk*, *soplex*, and *GemsFDTD*, respectively, when each application is executed alone. The x-axis of Fig. 1(a)–(c) represents the SLLC capacity as the number of ways, and the number of sets and block size are fixed to 4096 and 64 bytes, respectively. Also, the y-axis represents misses per kilo instructions (MPKI) of SLLC. To obtain each miss curve exactly, we perform simulations for all points in miss curve from 1-way (256 KB) to 16-way (4 MB).

[2]The allocation policy needs $N \times W$ model instances where N is the number of cores and W is the associativity of SLLC while model instances require both hardware and software overhead although the hardware overhead for each instance is negligible with only about 1% of LLC area [10].

TABLE I
Optimal Partition Allocations for Combinations of Two Applications

	xalancbmk:soplex	*xalancbmk:GemsFDTD*
LRU/conventional	13:3	13:3
LRU/WB-aware	13:3	14:2
DIP/conventional	10:6	12:4
DIP/WB-aware	10:6	14:2
PDP/conventional	11:5	12:4
PDP/WB-aware	11:5	13:3

A. Impact of Replacement Policy on Miss Curve

Fig. 1(a) illustrates the influence of a replacement policy on a miss curve, showing that DIP and PDP outperform the conventional replacement policy, LRU, in the entire region of miss curves. In Fig. 1(b), miss curve of *soplex* is also influenced by the choice of replacement policy.

Fig. 1(d) and (e) show composite miss curves for combination of two applications when executed concurrently and 4 MB SLLC with 16-way associativity is divided into two partitions by way-based cache partitioning. All possible partition allocations are represented in the x-axis of Fig. 1(d) and (e), and each partition allocation is shown as $x : y$ where x and y are the allocated partition size for the former and latter application, respectively. Table I represents the optimal partition allocation for each combination.

Fig. 1(d) shows the impact of replacement policy on cache partitioning clearly. Different replacement policies in optimally partitioned SLLC lead to different SLLC performance showing that DIP and PDP reduce 12.53% and 19.64% of MPKI, respectively, compared to LRU. Furthermore, all three

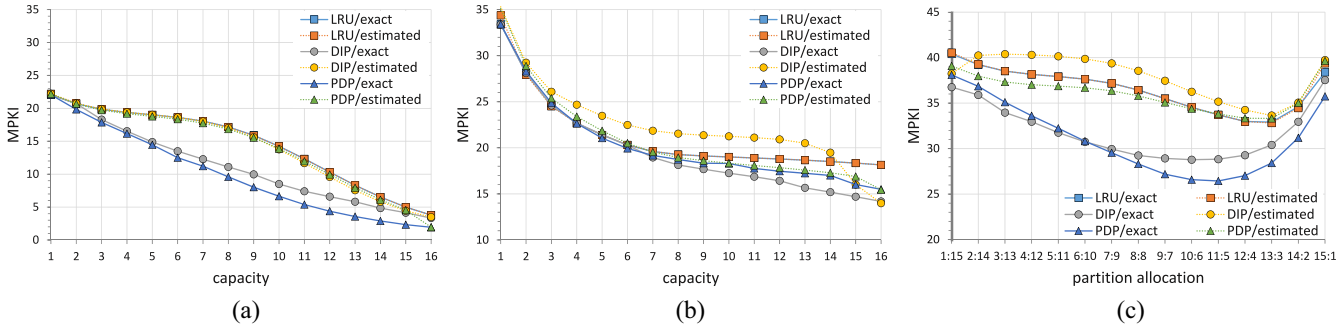


Fig. 2. Miss curve estimation from UMON [5]. (a) Estimated miss curves of *xalancbmk* from UMON. (b) Estimated miss curves of *soplex* from UMON. (c) Composite miss curves of *xalancbmk* and *soplex* from UMON.

replacement policies lead to different optimal partition allocations, and the optimal allocation of one replacement policy may not be optimal to other replacement policies. When partition allocation is fixed to the optimal allocation for LRU (13:3), partitioned SLLC incorporated with DIP and PDP show MPKI reduction of 7.59% and 13.65%, respectively, compared to that incorporated with LRU. This infers that cache partitioning method tailored for a specific replacement policy incurs suboptimal results when incorporated with other replacement policies.

B. Impact of Writeback Misses on Miss Curve

Fig. 1(c) shows the gap in MPKI between conventional miss curve and writeback-aware miss curve, and the gap is emphasized in low capacity case. In a noninclusive SLLC with small capacity, corresponding blocks on SLLC are already evicted with high probability when the upper-level cache sends writeback requests. This causes misses by writeback request, and latency of the misses can be hidden mostly thanks to memory-level parallelism. Nevertheless, the conventional miss curve cannot reflect this aspect and can be misleading. As the available capacity increases, the writeback-aware miss curve converges to conventional miss curve because SLLC can include most data in the upper-level caches. Fig. 1(b) also shows the similar gap in the case of SLLC with small SLLC capacity.

Fig. 1(e) shows the composite miss curves of *xalancbmk* and *GemsFDTD* from conventional and writeback-aware miss curves, and it also shows the optimal partition allocation with each composite miss curve. For all three replacement policies, the optimal partition allocation for writeback-aware miss curves grants more capacity to *xalancbmk* compared to that with conventional miss curves. Partitioned SLLC incorporated with LRU shows 8.05% difference in MPKIs at between the optimal partition allocations from conventional and writeback-aware miss curves. Also, partitioned SLLC with DIP and PDP show 4.13% and 4.02% difference, respectively. Thus, we can further improve SLLC performance as much as the differences, considering the fact that misses by writeback requests do not need to wait for responses from main memory and the latency of the misses can be hidden in most cases.

TABLE II
OPTIMAL PARTITION ALLOCATIONS FOR *xalancbmk+soplex* WITH EXACT AND ESTIMATED MISS CURVES

	with exact miss curves	with estimated miss curve by UMON
LRU	13:3	13:3
DIP	10:6	13:3
PDP	11:5	13:3

C. Impact of Miss Curve Estimation on Cache Partitioning

Fig. 2 shows exactly obtained miss curves (solid line) and estimated miss curves from UMON (dotted line) each combined with three different replacement policies. In Fig. 2(a) and (b), UMON can accurately estimate miss curves for LRU while it gives inaccurate miss curves for DIP and PDP. This is because UMON is designed to exploit LRU's stack property and DIP and PDP do not comply to the stack property.

Fig. 2(c) shows the combination of the estimated miss curves from UMON, and Table II represents the optimal partition allocations. In Fig. 2(c), miss curves from UMON can provide the optimal partition allocation for SLLC managed by LRU while giving suboptimal allocations for DIP and PDP. For partitioned SLLC managed by DIP and PDP, partition allocation shaped by UMON lead to 16.88% and 26.01% increase in MPKI compared to the optimal partition allocation from the exact miss curve estimation.

In this section, we observed how replacement policies and writeback miss affect miss curves. Conventional miss curve estimators can provide optimal results for given specific replacement policies, but they do not support cache partitioning incorporated with replacement policies other than preconfigured ones. To tackle the problem, we propose a novel miss curve estimation technique adaptable to arbitrary replacement policy with low overhead.

IV. REPLACEMENT POLICY ADAPTABLE CACHE PARTITIONING

A. Cache Partitioning Architecture

To clarify the scope of our proposed method, we first give a generalized cache partitioning architecture as shown

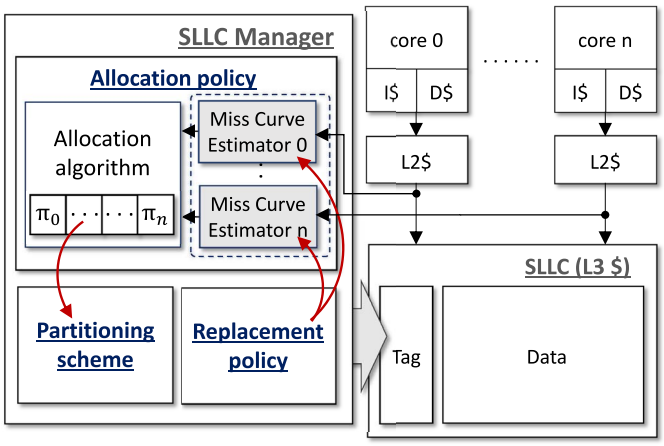


Fig. 3. Generalized cache partitioning architecture.

in Fig. 3. The generalized architecture consists of three main components.

1) An allocation policy which logically allocates an appropriate partition size, π_i, to each application, app$_i$.
2) A partitioning scheme which enforces π_i to app$_i$.
3) A replacement policy which manages SLLC contents of each partition with given π_i to app$_i$.

An allocation policy periodically recalculates the SLLC partition allocation for time interval t, $\Pi^t = \{\pi_1^t, \pi_2^t, \ldots, \pi_N^t\}$, such that performance of SLLC is maximized. Meanwhile, a partitioning scheme engages in the management of SLLC contents on every SLLC access with Π^t from allocation policy. It makes SLLC contents of each partition managed separately, and the management is performed with the assistance of a replacement policy.

Also, there exists dependency between an allocation policy and a replacement policy such that the allocation policy is strongly influenced by the replacement policy which can change workload pattern of applications on SLLC. The dependency prevents existing cache partitioning methods from being incorporated with an arbitrary replacement policy. For this reason, we focus on an allocation policy to make cache partitioning adaptive to any replacement policies.

An allocation policy consists of two components: 1) a *miss curve estimator* and 2) *allocation algorithm*. miss curve estimator generates an estimated miss curve, MC_i, for app$_i$, and allocation algorithm takes the estimated miss curves as inputs and returns the optimal partition allocation for the next time interval. Since SLLC behavior with respect to a replacement policy may be reflected on miss curves, miss curve estimation is the key component to allow an allocation policy adaptable to arbitrary replacement policy while allocation algorithm is just an exploration process with given miss curves and is independent of replacement policy. For our allocation policy, an allocation algorithm called *lookahead* algorithm [5] is adopted due to its effectiveness with reasonable complexity.

To summarize, the dependency between replacement policies and cache partitioning can be reflected on allocation decision of partition size to each application in form of miss curve. Therefore, we propose a novel miss curve estimation technique adaptable to arbitrary replacement policy.

B. Replacement Policy Adaptable Miss Curve Estimation

Unlike existing methods, we approximate miss curve without preliminary knowledge of given replacement policy. Instead, we only consider gradient of miss count and perform linear interpolation with a fixed number of sampling points, locations of which are changed adaptively. The key point is how many sampling points are required and how to dynamically determine the locations of sampling points within miss curves.

One of the premises of the proposed scheme is that allocation algorithm gives higher priority for partition allocation to the application with a larger gradient of miss count, thus it is important to find and to accurately estimate miss counts in a region of interest (ROI) of miss curve where the gradient varies most drastically. Another premise is that any miss curve is monotonically decreasing. Since larger cache size leads to less capacity misses for a given application, miss curves, a graph of miss counts versus cache size, have a monotonically decreasing shape. Two sampling points are used to find the lower-bound and upper-bound of miss curve. Since every miss curve has monotonically decreasing shape, miss count of $\pi_{\min}(\pi_{\max})$ is the maximum(minimum) value in miss curve where $\pi_{\min}$ and $\pi_{\max}$ denote the minimum and maximum partition size, respectively. Then, we can find the ROI through exploration using two sampling points with variable location: 1) allocation point and 2) guidance point. Allocation point is the actual allocation decision made by allocation algorithm while guidance point is used to guide allocation point toward ROI.

If we vertically split the miss curve from allocation point—chosen by incorporated allocation algorithm using estimated miss curve—into left segment($[\pi_{\min}, \pi_i]$) and right segment($[\pi_i, \pi_{\max}]$), the actual ROI would always reside on the side with larger gradient value, due to the monotonic nature of miss curves. Guidance point—where miss count is estimated in order to compute gradient value required in next time interval—is thus set to the median of the segment where the ROI would be, that is, the side with larger gradient value. By the end of a time interval, the miss count at guidance point is accurately estimated and is used to better estimate the miss curve using linear interpolation. Newly estimated miss curve will, in turn, be provided to the allocation algorithm to increase the odd of choosing allocation point closer to ROI. This way, allocation point converges toward ROI with time.

1) RME Architecture: Miss curve estimators usually use auxiliary tag directory (ATD) structures [5], [6]. ATD entries are updated upon every SLLC reference, and ATD reports whether the reference on the ATD is missed or not. This ATD operation is performed independently to real tag directory of SLLC. The purpose of ATD is to profile miss counts for a time interval with the given SLLC size and associativity, thus ATD is used together with miss counters.

Fig. 4 shows the architecture of RME. RME contains three ATDs: two fixed-sized ATDs (F-ATDs) and one variable-sized

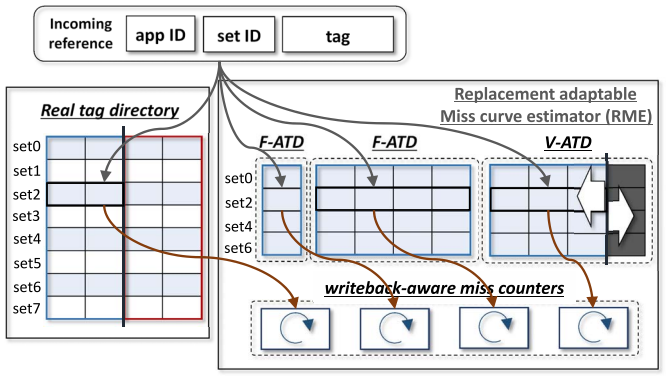


Fig. 4. Architecture of RME.

ATD (V-ATD). F-ATDs are used to estimate miss count for the minimum partition size ($\pi_{\min}$) and the maximum partition size ($\pi_{\max}$). The role of ATDs is to profile miss counts for the sampling points during a time interval. F-ATDs profile miss counts of $\pi_{\min}$ and $\pi_{\max}$ for the lower-bound and upper-bound of miss curve. On the other hand, V-ATD is used to profile miss count for a specific partition size ($\pi^t_{i-\text{var}}$) at given time interval t, and the V-ATD can be reconfigured to profile miss count of different partition size. The reconfiguration process is similar to real tag in SLLC, but it is less complex than the real tag in that V-ATD only targets a single application. Also, all three ATDs are managed by the same replacement policy used in the real tag. When a dynamic replacement policy is employed, parameters for replacement are dynamically adjusted for each application. In that case, the ATDs apply the parameters based on their miss count profiling results.

Our proposed RME also contains four miss counters for $\pi_{\min}$, $\pi_{\max}$, $\pi^t_{i-\text{var}}$, and the current partition size, π^t_i. Three miss counters for $\pi_{\min}$, $\pi_{\max}$, and $\pi^t_{i-\text{var}}$ are incremented upon a miss in the corresponding ATD, and miss counter for π_i is incremented upon a miss in the partition of the real tag. Thus, the miss counter for $\pi^t_{i-\text{var}}$ and π^t_i correspond to allocation point and guidance point, respectively. A key point of the miss counters is that they exclude misses by writebacks on their counting. This is because performance effects of line-fill and writeback are different [16]. Write-backs from the upper-level caches may not trigger main memory access immediately and their miss penalty can be hidden while misses caused by other commands always trigger line-fill, thereby triggering main memory read.

2) Algorithm for RMEs: The operation of RMEs consists of two main algorithms: 1) preallocation phase (Algorithm 1) and 2) post-allocation phase (Algorithm 2). In preallocation phase, each RME generates a miss curve of an application, MC^{t+1}_i, with four miss counters profiled during the time interval t. With generated miss curves for all RME, allocation algorithm mixes the miss curves and finds the optimal partition allocation for the next time interval, Π^{t+1}. After allocation, post-allocation phase of RME selects the location of sampling point for the next time interval, $\pi^{t+1}_{i-\text{var}}$ on a per-application basis.

Algorithm 1 Preallocation Phase for RMEs

Input: 4 sampled miss counts (for π_{min}, π_{max}, π^t_i, and π^t_{i-var}) of each application

Output: Estimated miss curves **to allocation algorithm**

1: **for** $1 \le i \le N$ **do** ▷ on per-application basis
2: curve-fitting immediate miss curve, IMC^t_i by linear interpolation of the 4 miss counters
3: **for** $1 \le j \le W$ **do**
4: $MC^{t+1}_{i,j} = \alpha \cdot IMC^t_{i,j} + (1-\alpha) \cdot MC^t_{i,j}$
5: **end for**
6: **end for**

Algorithm 2 Postallocation Phase for RMEs

Input: Π^{t+1} **from allocation algorithm**

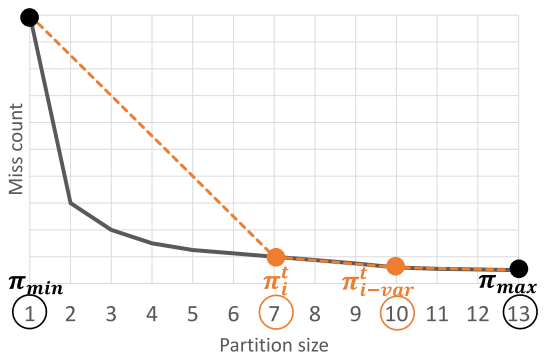
Output: π^{t+1}_{i-var} of each application

1: **for** $1 \le i \le N$ **do** ▷ on per-application basis
2: **if** $\pi^{t+1}_i < \pi_{min} + 2$ **then**
3: $\pi^{t+1}_{i-var} = (\pi^{t+1}_i + \pi_{max})/2$
4: **else if** $\pi^{t+1}_i > \pi_{max} - 2$ **then**
5: $\pi^{t+1}_{i-var} = (\pi_{min} + \pi^{t+1}_i)/2$
6: **else**
7: $\Delta_l = (MC^{t+1}_{i,\pi_{min}} - MC^{t+1}_{i,\pi_i})/(\pi^{t+1}_i - \pi_{min})$
8: $\Delta_r = (MC^{t+1}_{i,\pi_i} - MC^{t+1}_{i,\pi_{max}})/(\pi_{max} - \pi^{t+1}_i)$
9: **if** $\Delta_l < \Delta_r$ **then**
10: $\pi^{t+1}_{i-var} = (\pi^{t+1}_i + \pi_{max})/2$
11: **else**
12: $\pi^{t+1}_{i-var} = (\pi_{min} + \pi^{t+1}_i)/2$
13: **end if**
14: **end if**
15: reconfigure V-ATD for application i
16: **end for**

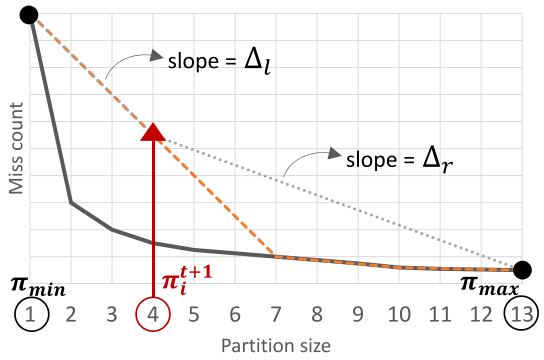
In Algorithm 1, miss curve for each application is estimated with the profiled miss counts during time interval t. An immediate miss curve of time interval t for each application is approximated by linear interpolation of the four miss counters (line 2), then exponential moving average (EMA) of miss curve is calculated to cope with the dynamic behavior of given application and to predict miss curve for the next time interval. The procedure of the prediction with EMA can be implemented easily with low overhead (lines 3–5).[3] Using estimated miss curves, the allocation algorithm allocates the proper partitions for time interval $t+1$.

In Algorithm 2, guidance point for time interval $t+1$, $\pi^{t+1}_{i-\text{var}}$, is calculated for each application. x-axis domain of miss curve is divided as the left side and right side on π^{t+1}_i, and then the gradient of the left side(right side), $\Delta_l(\Delta_r)$, are calculated (lines 7 and 8). If Δ_l is less than Δ_r, $\pi^{t+1}_{i-\text{var}}$ is set to the median between π^{t+1}_i and $\pi_{\max}$ (line 10). Otherwise, $\pi^{t+1}_{i-\text{var}}$ is set to

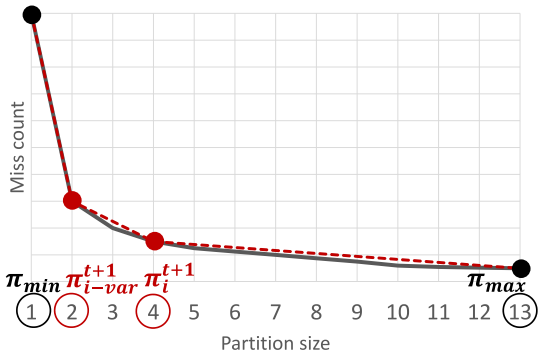
[3]Weighting coefficient α is set to 0.5 in this paper.



(a)



(b)



(c)

Fig. 5. Example of miss curve estimation by RME. (a) Exact miss curve (solid line) and approximated miss curve (dashed line) at the end of time interval t. (b) Exact miss curve (solid line) and approximated miss curve (dashed line) after calculating Π^{t+1}. (c) Exact miss curve (solid line) and approximated miss curve (dashed line) at the end of time interval $t+1$.

the median between $\pi_{\min}$ and π_i^{t+1} (line 12). As such, we can narrow down ROI in miss curve adaptively with V-ATD.

Fig. 5 illustrates an example of the miss curve estimation for application i. Fig. 5 shows the exact miss curve (solid line) and the most drastically changed region in the miss curve is at partition size 2. Assuming that π_i^t is 7 and $\pi_{i-\text{var}}^t$ is 10 for time interval t, then the miss counts for four sampled points [$\pi_{\min}(1)$, $\pi_i^t(7)$, $\pi_{i-\text{var}}^t(10)$, and $\pi_{\max}(13)$] are profiled during time interval t. Then, miss curve can be approximated by curve-fitting of four points [at $\pi_{\min}(1)$, $\pi_i^t(7)$, $\pi_{i-\text{var}}^t(10)$, and $\pi_{\max}(13)$] at the end of time interval $t+1$. The estimated

TABLE III
CACHE CONFIGURATION

L1 I/D	4-way 32KB, 64B line, LRU, 4 MSHRs, 4 writeback entries (for L1 D), 1-cycle latency
L2	8-way 128KB, 64B line, LRU, 8 MSHRs, 8 writeback entries, 6-cycle latency, non-inclusive
L3	4-way/core, 1MB/core, 64B line, 20 MSHRs, 8 writeback entries, 20-cycle latency, non-inclusive
Mem	4 banks, 200-cycle latency

TABLE IV
BENCHMARK CLASSIFICATION

category	type		benchmarks
A	capacity sensitive	replacement policy sensitive	*mcf*, *omnetpp*, *soplex*, *xalancbmk*, *sphinx3*
B		replacement policy insensitive	*bzip2*, *calculix*, *GemsFDTD*, *zeusmp*
C	capacity insensitive	streaming	*lbm*, *libquantum*, *milc*
D		CPU-bound	*astar*, *h264ref*, *hmmer*

miss curve is represented by a dashed line in Fig. 5(a). After the estimation of miss curve for each application, allocation algorithm mixes all miss curves and find the optimal partition allocation. Assume that π_i^{t+1} is chosen to be 4 for the next time interval $t+1$ after partition allocation. Then, $\pi_{i-\text{var}}^{t+1}$ is set to the median between $\pi_{\min}$ and π_i^{t+1} because Δ_l is greater than Δ_r, which is represented by a dotted line in Fig. 5(b). Thus, the locations of the sampled points are changed ($\pi_i : 7 \rightarrow 4$, $\pi_{i-\text{var}} : 10 \rightarrow 2$) and miss counts for the points are profiled during time interval $t+1$. To calculate partition allocation for time interval $t+2$, we again perform the linear interpolation of the four sampled points [$\pi_{\min}(1)$, $\pi_{i-\text{var}}^{t+1}(2)$, $\pi_i^{t+1}(4)$, and $\pi_{\max}(13)$] at the end of time interval $t+1$. Fig. 5(c) shows the approximated miss curve at the end of time interval $t+1$, and it accurately captures ROI in miss curve.

V. EXPERIMENTS

A. Experimental Methodology

We use a Pin [17] based trace-driven cache simulator for our SLLC simulation. Our target memory hierarchy consists of two private cache levels (L1 and L2) and a shared last level cache (L3). The configuration parameters are shown in Table III.

SPEC 2006 benchmark suite is used to construct multi-programmed workloads. The benchmarks are classified into four categories as shown in Table IV. Capacity-sensitive benchmarks are susceptible to performance loss with reduced SLLC partition size. The capacity-sensitive group is further categorized according to their sensitivity to a replacement policy. In the capacity-insensitive group, memory-intensive benchmarks are categorized as streaming and others are

TABLE V
BENCHMARK COMBINATIONS

mix4	quad-core workload	mix8	8-core workload
n4-1	mcf-xalancbmk-lbm-astar	n8-1	bzip2-omnetpp-xalancbmk-lbm-libquantum-milc-GemsFDTD-h264ref
n4-2	bzip2-omnetpp-lbm-astar	n8-2	bzip2-omnetpp-xalancbmk-lbm-libquantum-milc-GemsFDTD-hmmer
n4-3	bzip2-xalancbmk-lbm-astar	n8-3	bzip2-omnetpp-xalancbmk-mcf-lbm-libquantum-milc-GemsFDTD
n4-4	xalancbmk-lbm-libquantum-GemsFDTD	n8-4	bzip2-omnetpp-xalancbmk-zeusmp-lbm-libquantum-milc-GemsFDTD
n4-5	bzip2-omnetpp-lbm-libquantum	n8-5	bzip2-omnetpp-xalancbmk-zeusmp-lbm-libquantum-milc-hmmer
n4-6	bzip2-omnetpp-lbm-milc	n8-6	bzip2-omnetpp-xalancbmk-soplex-zeusmp-lbm-libquantum-milc
n4-7	bzip2-xalancbmk-lbm-libquantum	n8-7	calculix-omnetpp-xalancbmk-mcf-soplex-lbm-libquantum-milc
n4-8	omnetpp-lbm-milc-GemsFDTD	n8-8	calculix-xalancbmk-mcf-soplex-sphinx3-lbm-libquantum-milc
n4-9	calculix-xalancbmk-lbm-milc	n8-9	calculix-GemsFDTD-xalancbmk-mcf-soplex-lbm-libquantum-milc
n4-10	mcf-omnetpp-lbm-milc	n8-10	bzip2-calculix-GemsFDTD-omnetpp-xalancbmk-lbm-libquantum-milc
n4-11	soplex-xalancbmk-lbm-astar	n8-11	calculix-GemsFDTD-zeusmp-xalancbmk-mcf-lbm-libquantum-milc
n4-12	omnetpp-soplex-lbm-astar	n8-12	calculix-GemsFDTD-xalancbmk-mcf-lbm-libquantum-milc-hmmer
n4-13	mcf-omnetpp-lbm-libquantum	n8-13	bzip2-omnetpp-xalancbmk-mcf-lbm-libquantum-milc-hmmer
n4-14	xalancbmk-mcf-sphinx3-lbm	n8-14	omnetpp-xalancbmk-GemsFDTD-lbm-libquantum-milc-astar-h264ref
n4-15	mcf-omnetpp-lbm-astar	n8-15	omnetpp-xalancbmk-bzip2-lbm-libquantum-milc-hmmer-h264ref
n4-16	mcf-xalancbmk-lbm-libquantum	n8-16	soplex-xalancbmk-GemsFDTD-lbm-libquantum-milc-astar-h264ref

categorized as CPU-bound benchmarks. Table V shows constructed multiprogrammed workloads for quad-core and 8-core systems.

We run each benchmark 250 MB instructions after bypassing 10 B instructions. If a benchmark finishes 250 MB instructions while others are still running, the benchmark continues to run until all benchmarks complete 250 MB instructions. The simulator returns the simulation results of only first 250 MB instructions for each benchmark after the simulation is finished. This simulation methodology is similar to [6].

Two performance metrics are used as

$$\text{throughput} = \sum_{i=1}^{N} \text{IPC}_i \tag{1}$$

$$\text{fair speedup} = \frac{N}{\sum_{i=1}^{N} (\text{Single IPC}_i / \text{IPC}_i)}. \tag{2}$$

In (1) and (2), IPC_i denotes the number of executed instructions per cycle in core i and Single IPC_i denotes the number of executed instructions per cycle in core i when core i is run alone with LRU policy. Throughput in (1) indicates system utilization and fair speedup in (2) balances both performance and fairness among cores [18]. In (2), $\text{IPC}_i/\text{Single IPC}_i$ denotes speedup of core i, and fair speedup is the harmonic mean of the speedups of individual cores. Since the harmonic mean represents the average of the speedups, fair speedup can reflect the fairness among cores as well as performance of each core.

To show its adaptability to replacement policies, RCP is run in conjunction with multiple replacement policies: LRU, DGIPPR [12],[4] LFU [13], TA-DIP [1], TA-DRRIP [2], and PDP [3].[5,6] The combinations are represented as RCP+LRU, RCP+DGIPPR, RCP+LFU, RCP+TA-DIP, RCP+TA-DRRIP, and RCP+PDP, respectively. Then the results are compared with two cache partitioning schemes: UCP [5] and CLU [6].[7] Since way-based partitioning scheme is used in both UCP and CLU, RCP also adopts way-based partitioning scheme in these experiments for fair comparison. In the experiments, partition allocation is periodically recalculated every five million cycles for all cache partitioning schemes.

In this section, we first demonstrate the efficiency of RME by showing the accuracy of RME versus an exhaustive miss curve estimator which estimates miss counts for every available partition size. Then, we analyze the adaptability of RCP and the effectiveness of writeback-aware miss curve. Finally, we show the impact of RME on cache partitioning and the synergy of replacement policy and cache partitioning using RCP.

B. Accuracy of Estimated Miss Curves From RME

To reduce miss curve estimation overhead, RME tries to capture ROI in miss curve through interaction with allocation algorithm and to accurately estimate the ROI rather than the entire region in miss curve. Fig. 6 shows how accurately RME finds and estimates ROI in miss curves. In Fig. 6, accuracy of estimated miss curves from RMEs with quad-core workloads is compared to exhaustive miss curves within ROIs of various sizes with π_i as the center. An exhaustive miss curve for a single application is obtained by estimating miss counts of all points in a miss curve exhaustively using W ATDs for W-way

[4]4-DGIPPR is used with multiset dueling. We use the following four IPVs for any SLLC partition of k-way ($1 \le k \le W$) : $\text{IPV}_0 = [0, \ldots, 0, 0]$, $\text{IPV}_1 = [0, \ldots, 0, k-1]$, $\text{IPV}_2 = [0, \ldots, k/2, k-1]$, and IPV_3 which is a random vector for a partition of k-way. The purpose of this policy is to demonstrate the adaptability of RCP with various types of replacement policies. For the best performance of DGIPPR, the optimal IPVs for every available SLLC partition size need to be evolved using genetic algorithm.

[5]In TA-DIP and TA-DRRIP, SDMs contain 32 sets and PSEL counters are 10-bit. Bimodal throttle parameter of TA-DIP and TA-DRRIP is 1/32. 2-bit RRPV registers are used in TA-DRRIP. $d_{\max}$ of PDP is 256.

[6]PDP utilizes a by-pass mechanism for a noninclusive cache. In this paper, we exclude by-pass mechanism from PDP to focus on the impact of replacement policy and cache partitioning on SLLC performance.

[7]Set sampling rate of UCP, CLU, and RCP is 1/64.

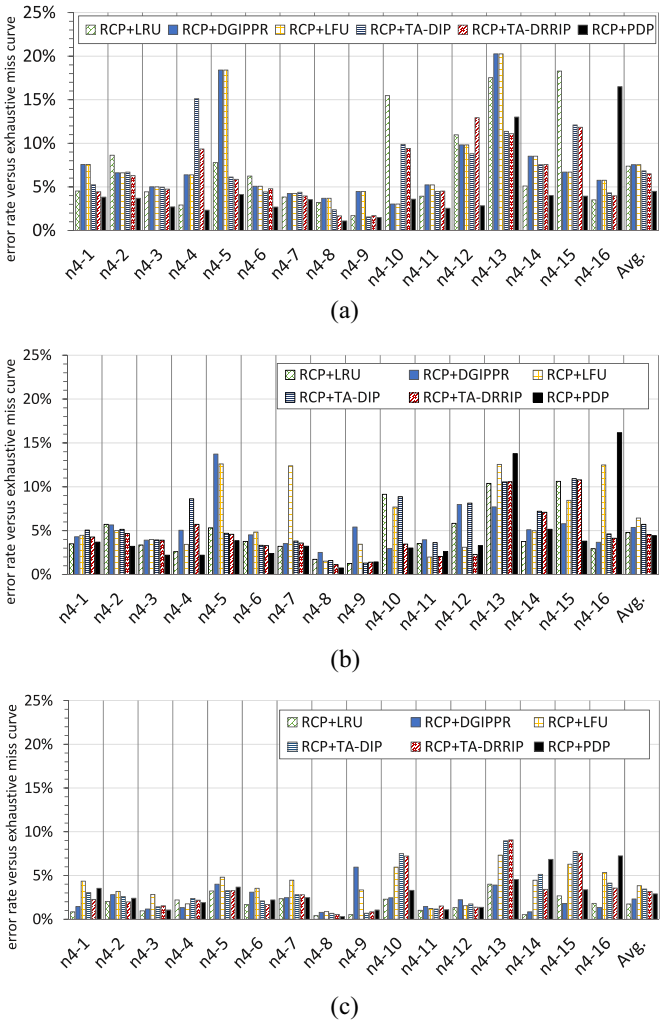


Fig. 6. Error rate of RME compared to the exhaustive miss curve estimation. (a) Error rate of RME in the ROI of size 5 with π_i as the center. (b) Error rate of RME in the ROI of size 3 with π_i as the center. (c) Error rate of RME in the ROI of size 1 (at π_i).

SLLC, while RME approximates the miss curve by performing linear interpolation with four sampled points of the miss curve to reduce the estimation overhead.

For all six replacement policies, Fig. 6(a) shows 6.98% of average error rate when incorporated with RME compared to the exhaustive miss curve in the ROI of size 5 with π_i as the center, and Fig. 6(b) shows 5.22% of average error rate of RME compared to the exhaustive miss curve in the ROI of size 3 with π_i as the center. In Fig. 6(c), 2.90% of an average error rate of RME is shown compared to the exhaustive miss curve at the single point, π_i.

These results show that average error rate is reduced as the size of the ROIs decreases, and it is worth noting that the deviation for workloads is also reduced as the size decreases. Nonetheless, in Fig. 6(a), some workloads (*n*4-5, *n*4-13, and *n*4-15) show high error rates. The main reason is that π_i is distant from $\pi_{i-\text{var}}$, preventing $\pi_{i-\text{var}}$ from approaching the ROI. This is because π_i is close to $\pi_{\min}$ or $\pi_{\max}$, thus $\pi_{i-\text{var}}$ is located nearby the center of miss curve $((\pi_{\min} + \pi_{\max})/2)$ by Algorithm 1 (from lines 2–5). In these cases, the accuracy

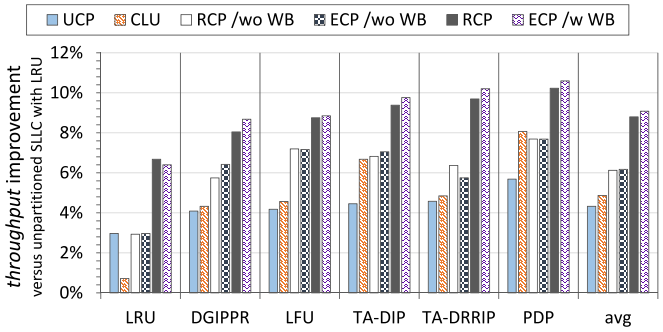


Fig. 7. Throughput improvement of each cache partitioning scheme under multiple replacement policies, over unpartitioned SLLC managed by LRU, averaged from 16 quad-core workloads (from *n*4-1 to *n*4-16).

at π_i is guaranteed because RME profiles miss counts at $\pi_{\min}$ and $\pi_{\max}$ in every time interval although the workloads show high error rates in the region of size 5. This way, RME can approach ROIs and can thus accurately estimate miss counts in the ROIs although ROIs are changed according to dynamically changed workloads over time.

C. Performance Impact by Adaptability to Replacement Policies

To show the effectiveness of the adaptability of RCP to an arbitrary replacement policy, we observe how system performance is affected when cache partitioning schemes are incorporated with various replacement policies. Fig. 7 shows the throughput improvement of each cache partitioning scheme under multiple replacement policies, over unpartitioned SLLC managed by LRU. Each bar in Fig. 7 shows an averaged value from 16 quad-core workloads, which are described in Table V. The cache partitioning schemes compared in Fig. 7 are listed as below.

1) *UCP:* Based on miss curve estimation specialized to LRU from UMON, which cannot obtain writeback-aware miss curves due to its structural limitation.[8]
2) *CLU:* Based on both UMON for LRU and LMON for BIP. CLU cannot also take account of writeback misses on its miss curve estimation due to UMON's limitation.
3) *RCP /wo WB:* Based on RME without writeback-aware miss curve.
4) *ECP /wo WB:* Based on exhaustive miss curve estimation without writeback-aware miss curve.
5) *RCP:* Based on RME with writeback-aware miss curve, the proposed method.
6) *ECP /w WB:* Based on exhaustive miss curve estimation with writeback-aware miss curve.

Note that CLU in this experiment only provides partition allocation under arbitrary replacement policy while the original CLU have an ability to select the proper replacement policy between LRU and BIP. Also, ECP uses EMA to predict miss

[8]In UMON, which uses a single ATD with the same configuration with a target SLLC, writeback misses are always filled in the LRU position, thus intrapartition interferences by writeback misses in a small-sized partition cannot be reproduced during miss curve tracking of UMON.

curves for the next interval like RCP. Because of misprediction of EMA, ECP cannot always guarantee the optimality of miss curve estimation although it shows the best performance improvement in most cases.

Fig. 7 shows that cache partitioning schemes under high-performance replacement policies deliver further performance improvement over those under LRU. The reason is that the replacement policies can further reduce intra-application interference over LRU with given capacity size for each application. However, the results are suboptimal in UCP and CLU since they are optimized for certain replacement policies and thus may not provide optimal partition allocation to SLLC managed by other replacement policies.

To focus on the adaptability to replacement policies, RCP /wo WB is compared with other cache partitioning schemes, which excludes the effect of writeback-aware miss curve estimation. In Fig. 7, RCP /wo WB can provide the near-optimal partition allocation because RME can estimate miss curves accurately regardless of incorporated replacement policy. Similar to UCP and CLU, RCP /wo WB incorporated with a high-performance replacement policy shows further improvement over RCP /wo WB with LRU as shown in Fig. 7. However, the performance improvement in RCP /wo WB is larger than UCP and CLU. The results imply that RCP /wo WB provides more optimal partition allocation than UCP and CLU under arbitrary replacement policy. The results imply that under most replacement policies, RCP /wo WB provides more optimal partition allocations than UCP and CLU. On average for all six replacement policies, RCP /wo WB provides 1.80% and 1.26% higher throughput than UCP and CLU, respectively, while it shows under 0.01% of difference with ECP /wo WB which gives near-optimal partition allocations for arbitrary replacement policy. Therefore, replacement policy adaptability of RCP can lead to further improvement of system performance by generating near-optimal partition allocations under arbitrary replacement policy.

D. Performance Impact by Writeback-Aware Miss Curve Estimation

Let us revisit Fig. 7 to demonstrate the performance impact of writeback-aware miss curves on cache partitioning. To show the impact of writeback-aware miss curves, we compare RCP with RCP /wo WB and we set ECP /w WB and ECP /wo WB as references for this comparison. In Fig. 7, RCP provides 2.67% higher throughput over RCP /wo WB on average while ECP /w WB provides 2.91% higher throughput over ECP /wo WB on average. Therefore, the comparison shows that RCP can almost fully exploit the asymmetry of different miss types to maximize the system performance.

In addition, Fig. 8 shows the influence of writeback-aware miss curve on partition allocation for each application to explain the reason for performance improvement through writeback-aware miss curve. Fig. 8(a) gives the percentage of writeback accesses on total SLLC accesses for each application, and we obtain this statistics from simulation with SLLC managed by UCP and LRU. In Fig. 8(b), partition allocation determined by RCP+LRU based on writeback-aware miss

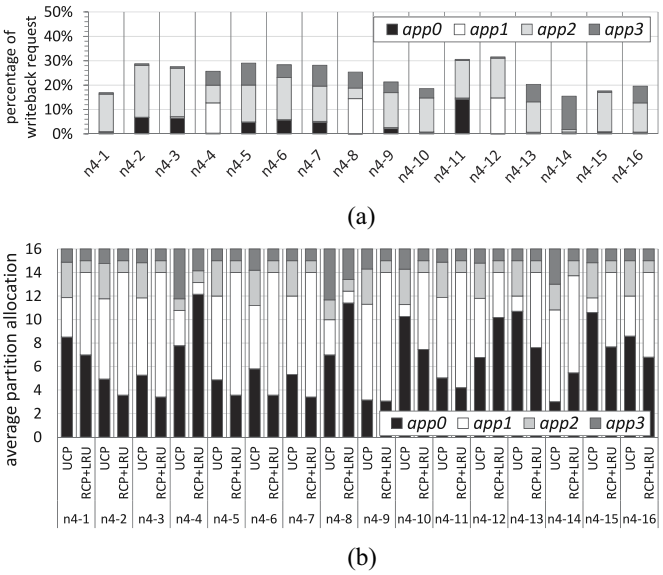


Fig. 8. Effect of writeback-aware miss curves on partition allocation for quad-core workloads consisting of four applications. (a) Percentage of writeback accesses on total SLLC accesses for each application, app_i. (b) Comparison of average partition allocation by UCP and that by RCP+LRU.

curve estimation reduces partition size of applications with a large percentage of writebacks while it increases partition size of other applications in comparison to partition allocation by UCP. Since misses caused by writeback requests from the upper-level caches do not need to send responses to CPUs, the latency of the writeback misses on SLLC can be hidden. Thus, it is unnecessary to allocate capacity for the writeback misses. As shown in Fig. 8(b), The redundant capacity allocation can be prevented by writeback-aware miss curves, and the capacity is allocated to applications which can utilize allocated capacity more effectively.

E. Total Performance Improvement

Finally, the combinations of RCP and multiple replacement policies are compared with the existing cache partitioning methods in terms of system performance. Fig. 9 shows performance improvement over unpartitioned SLLC managed by LRU with quad-core workloads. Fig. 9 shows the efficiency of RCP as a cache partitioning scheme by comparing with the original UCP incorporated with LRU and CLU incorporated with both LRU and BIP. RCP+LRU is compared with UCP which is specialized to LRU and RCP+TA-DIP is also compared with CLU whose target replacement policy is similar with TA-DIP. In Fig. 9(a), RCP+LRU provides on average 3.71% throughput improvement over UCP while RCP+TA-DIP provides on average 2.55% throughput improvement over CLU. With the same (or similar) replacement policy, RCP performs partition allocation more efficiently than UCP and CLU, and the main source of the improvement is writeback-aware miss curve estimation. UCP and CLU do not consider the effect of writeback misses in their miss curve estimation, thus they allocate unnecessarily large partition to applications with a large number of writeback accesses. Fig. 9(b) also supports the trend. In Fig. 9(b), RCP+LRU provides on average 3.84%

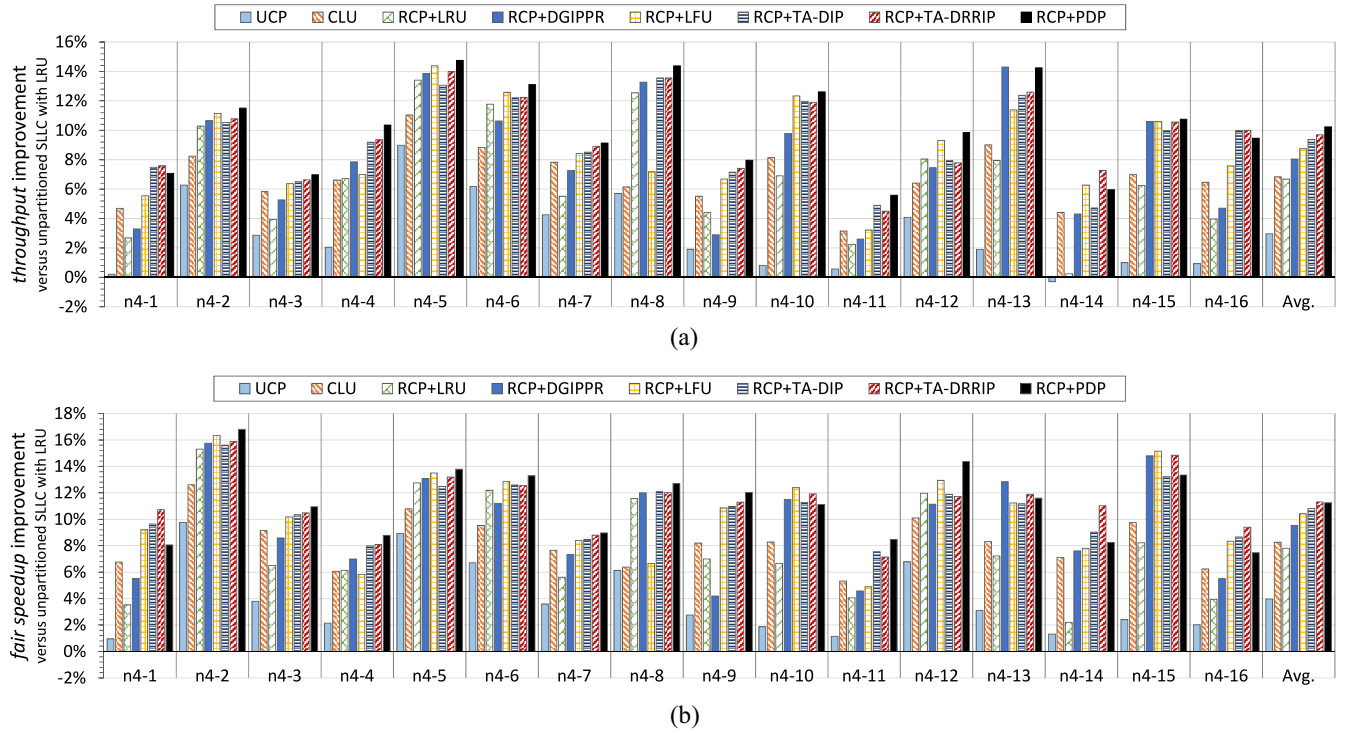


Fig. 9. Performance improvement versus unpartitioned SLLC managed by LRU with quad-core workloads. (a) Throughput improvement versus unpartitioned SLLC managed by LRU. (b) Fair speedup improvement versus unpartitioned SLLC managed by LRU.

fair speedup improvement over UCP while RCP+TA-DIP provides 2.55% fair speedup improvement over CLU.

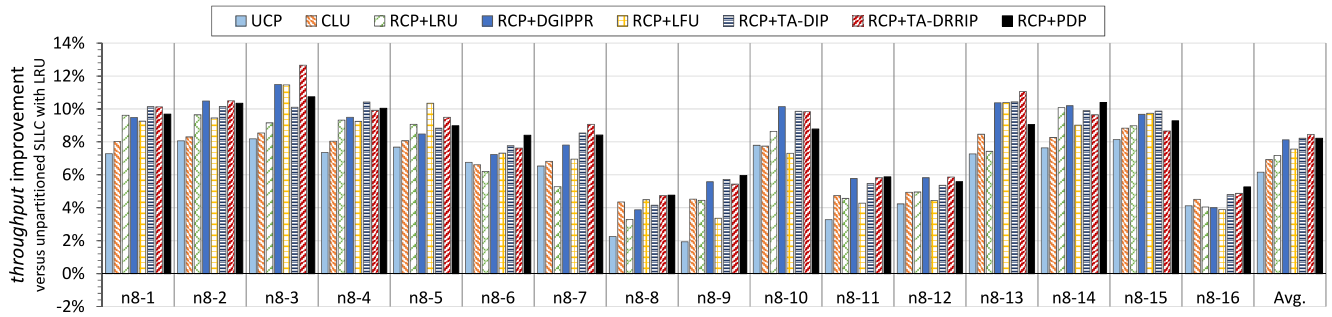
In addition, Fig. 9 also shows performance gain of RCP according to a replacement policy. In Fig. 9(a), RCP+DGIPPR provides up to 6.35% (on average 1.37%) higher throughput while RCP+LFU provides up to 6.03% (on average 2.07%) higher throughput, compared with RCP+LRU. Meanwhile, RCP+TA-DIP provides up to 6.02% (on average 2.70%) higher throughput while RCP+TA-DRRIP provides up to 7.03% (on average 3.01%) higher throughput compared with RCP+LRU. Also, RCP+PDP provides up to 6.30% (on average 3.55%) higher throughput compared to RCP+LRU. Since replacement policies, especially PDP, have the indirect partitioning effect, performance improvements by cache partitioning and replacement policy are partially overlapped. Nevertheless, cache partitioning with high-performance replacement policy outperforms cache partitioning with LRU. The main reason for the performance gap comes from efficiency of replacement policies to reduce intra-application interferences. LRU in SLLC cannot cope with access patterns with low locality whose reuse distance is larger than the associativity of the SLLC, while other policies can handle these so called thrashing access patterns. Finally, the best combination, RCP+PDP, shows up to 8.23% (on average 3.40%) throughput improvement compared to the best technique between UCP and CLU. Fair speedup improvement is similar to throughput improvement as shown in Fig. 9(b). RCP+DGIPPR provides up to 6.58% (on average 1.73%) higher fair speedup, and RCP+LFU provides up to 6.93% (on average 2.61%) higher fair speedup compared to RCP+LRU. Also, RCP+TA-DIP provides up to 6.82% (on average 3.01%) higher fair speedup, and RCP+TA-DRRIP provides up to 8.81% (on average 3.51%) higher fair speedup compared to RCP+LRU. RCP+PDP provides up to 6.04% (on average 3.44%) higher fair speedup compared to RCP+LRU. Thus, the results represent that the fairness among applications is well maintained while the throughput is improved by RCP incorporated with high-performance replacement policies.

Fig. 10 shows performance improvement over unpartitioned SLLC managed by LRU with 8-core workloads. In Fig. 10(a), RCP+LRU provides up to 2.51% (on average 1.01%) throughput improvement over UCP, and RCP+TA-DIP provides up to 2.39% (on average 1.30%) throughput improvement over CLU. Similarly, RCP+LRU provides up to 2.73% (on average 0.97%) fair speedup improvement over UCP, and RCP+TA-DIP provides up to 2.41% (on average 1.30%) fair speedup improvement over CLU in Fig. 10(b). Also, RCP+PDP shows up to 3.14% (on average 1.05%) throughput improvement over RCP+LRU, showing up to 2.96% (on average 1.27%) fair speedup improvement over RCP+LRU. The trend shown in Fig. 9 is still valid in Fig. 10.

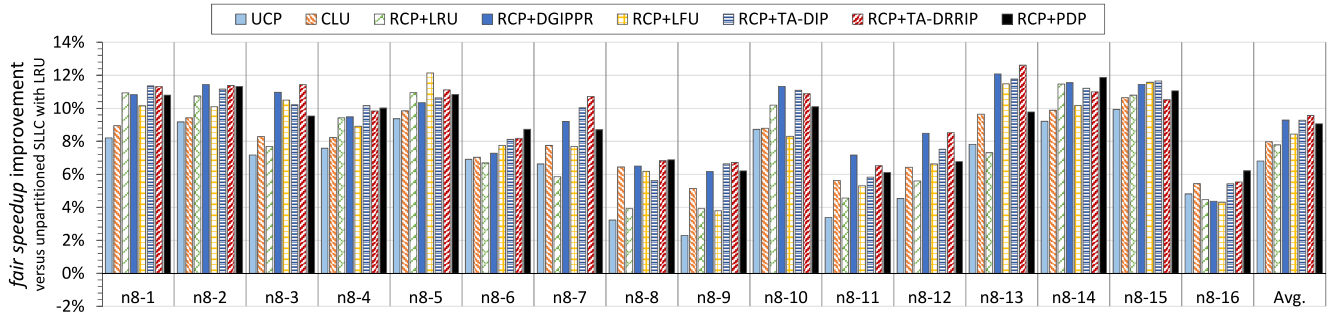
Therefore, the combination of RCP and a high-performance replacement policy outperforms existing cache partitioning methods which are tailored to conventional replacement policies by being able to provide the high-performance replacement policy an opportunity to further reduce intra-application interference.

F. Hardware Overhead of RME

Both hardware-based miss curve estimation and allocation algorithm for partition allocation require runtime performance



(a)



(b)

Fig. 10. Performance improvement versus unpartitioned SLLC managed by LRU with 8-core workloads. (a) Throughput improvement versus unpartitioned SLLC managed by LRU. (b) Fair speedup improvement versus unpartitioned SLLC managed by LRU.

TABLE VI
STORAGE OVERHEAD FOR MISS CURVE ESTIMATION OF A SINGLE APPLICATION

	# of ATDs	# of ATD entries / sampled set	# of counters
UCP	1	W	W
CLU	$1 + log_2 W$	$W + \sum_{k=1}^{log_2 W} 2^k = 3W - 1$	$W + log_2 W$
RCP	3	$\pi_{min} + 2 \cdot \pi_{max} = 2W - 2N + 3$	$4 + \pi_{max} = W - N + 5$
ECP	W	$\sum_{k=1}^{W} k = W(W+1)/2$	W

overheads. As described in [6], however, the overheads have a marginal impact on the system performance. Operations of RME and the allocation algorithm are performed in the background of normal SLLC operations, thus the performance impact of them can be hidden; ATDs of RME update miss counters on SLLC access to the sampled set, and this profiling is performed in parallel with normal SLLC operations. Miss curve estimation with the profiled miss counts and decision making for partition allocation are performed at the end of every time interval, and the calculations can be also performed in the background of normal SLLC operations. Thus, RCP do not affect the critical paths of normal SLLC operations.

In contrast to the performance overhead, hardware overhead is inevitable for miss curve estimation. Especially, storage overhead of ATDs and counters are dominant [5]. In Table VI, we compare the overhead of RME and other methods: UCP based on UMON for LRU, CLU based on UMON and LMON for BIP, and ECP with ATDs for every available partition sizes. Since dynamic set sampling [19] is used to reduce ATD overhead and the number of sampled set is a variable factor, we compare the number of ATD entries of ATDs per set rather than the total number of ATD entries in ATDs. The overhead of ECP increases exponentially with associativity while the overheads of other methods increase linearly. Compared to CLU based on UMON and LMON which are lightweight miss curve estimators, RCP requires about 2/3 the overhead of CLU. Thus, the overhead of RCP is competitive to state-of-art low-overhead methods, while providing a better performance. Note that the overhead shown in Table VI is for a single application and the total overhead for multiprogrammed workloads linearly increases with the number of applications.

Assume that the size of ATD entry is reduced to 10 bits by using hash function [6] and the number of sampled sets in each ATD is 64. For 4 MB SLLC with 16-way shared by four applications, storage overhead of RME for a single application is about 2 KB (0.05% of SLLC) and the total storage overhead for four applications is about 9 KB (0.21%) while that of ECP is 43 KB (0.98%). For 8MB SLLC with 32-way shared by eight applications, the storage overhead for a single application is about 4 KB (0.05% of SLLC) and the total overhead is about 33 KB (0.38%) while that of ECP is 331 KB (3.80%). Thus, RCP is an efficient cache partitioning method, providing performance comparable to ECP while requiring much less storage overhead.

VI. Conclusion

While high-performance replacement policies have been actively proposed, they cannot be incorporated with most existing cache partitioning methods which are dedicated to a specific conventional replacement policy. This paper proposes an RME which enables cache partitioning to be incorporated with arbitrary replacement policy with low overhead. Experimental results show that RCP can be effectively configured with high-performance replacement policies effectively, and RCP with high-performance replacement policy outperforms the existing cache partitioning schemes.

References

[1] A. Jaleel *et al.*, "Adaptive insertion policies for managing shared caches," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Tech.*, 2008, pp. 208–219.

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

[3] N. Duong *et al.*, "Improving cache management policies using dynamic reuse distances," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, Vancouver, BC, Canada, 2012, pp. 389–400.

[4] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 57–68, 2011.

[5] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2006, pp. 423–432.

[6] D. Zhan, H. Jiang, and S. C. Seth, "Clu: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches," *IEEE Trans. Comput.*, vol. 63, no. 7, pp. 1656–1667, Jul. 2014.

[7] H. Kasture and D. Sanchez, "Ubik: Efficient cache sharing with strict qos for latency-critical workloads," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 729–742, 2014.

[8] Y. Xie and G. H. Loh, "PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 174–183, 2009.

[9] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, New Orleans, LA, USA, 2012, pp. 1–12.

[10] N. Beckmann and D. Sanchez, "Modeling cache performance beyond LRU," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Barcelona, Spain, 2016, pp. 225–236.

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

[12] D. A. Jiménez, "Insertion and promotion for tree-based PseudoLRU last-level caches," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit.*, Davis, CA, USA, 2013, pp. 284–296.

[13] G. Bilardi, K. Ekanadham, and P. Pattnaik, "Efficient stack distance computation for priority replacement policies," in *Proc. 8th ACM Int. Conf. Comput. Front.*, 2011, p. 2.

[14] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970.

[15] H. Cook *et al.*, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 308–319, 2013.

[16] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutlu, and D. A. Jiménez, "Improving cache performance using read-write partitioning," in *Proc. IEEE Int. Symp. High Performance Comput. Archit. (HPCA)*, Orlando, Florida, USA, 2014, pp. 452–463.

[17] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 190–200, 2005.

[18] K. Luo, J. Gummaraju, and M. Franklin, "Balancing throughput and fairness in SMT processors," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2001, pp. 164–171.

[19] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for MLP-aware cache replacement," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 167–178, 2006.

Byunghoon Lee received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2010, where he is currently pursuing the Ph.D. degree in electrical and electronic engineering.

His current research interests include system architecture and low power design.

Kwangsu Kim received the B.S. degree from Yonsei University, Seoul, South Korea, in 2014, where he is currently pursuing the master's degree and the Ph.D. degree in electrical and electronic engineering.

His current research interests include memory hierarchy, near-data processing, and ultralow power computing. He did an internship in Qualcomm Korea, Seoul, in 2012.

Eui-Young Chung (S'99–M'06) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2002.

From 1990 to 2005, he was a Principal Engineer with SoC Research and Development Center, Samsung Electronics, Yongin, South Korea. He is currently a Professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul. His current research interests include system architecture and very large scale integration design, including all aspects of computer-aided design with the special emphasis on low-power applications and flash memory applications.