IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 16, 2021, accepted November 2, 2021, date of publication November 8, 2021,
date of current version November 29, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3126739

Energy-Efficient Shared Cache Using Way Prediction Based on Way Access Dominance Detection

YUN-SEOK OH, (Student Member, IEEE), AND EUI-YOUNG CHUNG, (Member, IEEE)
School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Eui-Young Chung (eychung@yonsei.ac.kr)

This work was supported in part by the Samsung Electronics Company, Ltd., Hwaseong, South Korea; in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Korean Government [Ministry of Science and ICT (MSIT)] under Grant 2021-0-00754; and in part by the Software Systems for AI Semiconductor Design.

ABSTRACT To meet the performance demands of chip multiprocessors, chip designers have increased the capacity and hierarchy of cache memories. Accordingly, a shared lower-level cache reduces conflict misses by adopting a multi-way set-associative structure with high associativity. This structure allows fast access because it allows access to all the ways in the cache set in parallel. However, it consumes a large amount of dynamic energy. Therefore, various schemes have been proposed to increase the energy efficiency of the cache memory. These schemes use *way prediction* or *partial comparison* to reduce unnecessary way access. This paper proposes a *way prediction* algorithm suitable for a shared second-level cache with high associativity. This algorithm is based on real-time *way access dominance detection* (*WADD*). Through this detection, the proposed algorithm can determine the number and location of *way candidates* suitable for each *partial access pattern* among the fragmented access patterns owing to the first-level cache replacement policy and intermingled accesses by multiple cores. Through this process, the proposed algorithm can implement an efficient *way prediction*. Simulation results show that the *WADD* exhibits the highest energy efficiency among the comparison groups, thus reducing the energy-delay product by 13.5% compared with the conventional cache without *way prediction*. This result is achieved by reducing the *way prediction* penalty through fast detection and high prediction accuracy.

INDEX TERMS Cache memory, computer architecture, energy efficiency, multiprocessing systems, way prediction.

I. INTRODUCTION

The demands for workloads with a large working set size such as advanced applications such as 3D graphics-based user interfaces or cloud-based digital services are growing in the recent industry. Due to these market demands for high performance, most platforms, including servers and mobile, have adopted chip multi-processors (CMPs). A CMP requires high bandwidth to meet the required performance. Cache memories with increased capacity and hierarchy are also required to minimize performance bottlenecks. For example, the M1 chip announced by Apple in 2020 features a shared 12MB second-level (L2) cache for four high-performance cores and a shared 4MB L2 cache for four high-efficiency cores. Accordingly, the shared L2 cache applies high associativity with a large capacity to reduce conflict misses [1]–[4]. However, these features make cache memory one of the most highly power-consuming devices in modern processors. According to Intel's report [5], the energy consumed by a cache memory is between 12% and 45% of the total energy consumed, depending on the computation amount of the application. Therefore, maximizing cache energy efficiency is a crucial challenge for chip designers.

A commonly used cache architecture is multi-way set-associative. Multi-way set-associative caches require less search effort than fully associative caches. Moreover, they sustain less data contention than direct-mapped caches. In a set-associative cache, finding a way with the necessary data, the tag array of all ways in the cache set in parallel should be accessed and searched. Since the requested data exist in only one specific way, a high-associativity cache is relatively inefficient in dynamic power consumption [6]–[8].

The associate editor coordinating the review of this manuscript and approving it for publication was Libo Huang.

Several *way prediction* schemes for improving cache energy efficiency have been proposed in the literature [9]–[20]. *Way prediction* schemes predict the *way candidates* based on previous cache accesses, allowing the cache to access the *way candidates* only. Thus, these schemes reduce the dynamic power consumption because they reduce access to unnecessary ways. However, the cache suffers from delay and power penalties if the prediction is inaccurate because it reaccesses the other ways to find the correct way. Therefore, high accuracy is essential.

Most *way prediction* techniques utilize the *recently-based locality* property [9]–[14]. However, in the second-level (L2) cache, the first-level (L1) cache replacement policy weakens the locality property. Additionally, L2 caches have a higher degree of associativity. Because of the combination of fragmented access patterns and high associativity, the above schemes are inefficient for a high-associativity L2 cache. *Way determination* schemes [16]–[19] detect the regularity of the cache access pattern in this environment. However, for a shared L2 cache in CMPs, each core's fragmented patterns are intermingled. Therefore, various reference intervals due to fragmented patterns reduce the *way prediction* accuracy.

Schemes for pre-determining cache misses have also been studied [21]–[25] as an alternative to using *way prediction* to reduce the number of *way candidates* for the subsequent cache access. These schemes use *partial tag comparison* [21]–[23] or the *modified Bloom filter* [24], [25]. They can significantly reduce unnecessary accesses because they detect non-selected ways in advance and halt access to the data array of non-selected ways. However, they require additional hardware resources for implementation and often have false positives. Hence, their efficiency is low in an L2 cache with a large capacity and high associativity characteristics.

This paper proposes a *way prediction* algorithm based on the *way access dominance detection* (*WADD*) for a shared L2 cache with high associativity. This algorithm achieves energy-efficient *way prediction* by maintaining high prediction accuracy while using relatively few *way candidates*. We call the concentration of access on specific ways *way access dominance*, and call the specific ways *dominant ways*. The proposed algorithm is motivated by occurring this *way access dominance* as the workload progresses. Thus, the *WADD* allocates small-sized counters for each way in the set. These counters, updated on every cache access, can detect the *dominant ways* in real-time. The *WADD* uses this detection result to maximize energy efficiency by selecting the appropriate number of *way candidates* for cases where the *way prediction* is necessary. The additional overhead required to support this operation is only approximately 1% of the target cache size. In this paper, the following contributions are made:

- Since the proposed algorithm continuously detects the *dominant ways*, it can quickly discover cache access pattern changes. Consequently, it is possible to respond to fragmented patterns in time with the L1 cache replacement policy, thus reducing the *way prediction* penalty in the L2 cache.
- The proposed algorithm identifies the cores accessing each way and performs *dominance detection* for each core separately. Since this algorithm handles each core's access patterns separately, it can perform the *way prediction* unaffected, even in a shared cache environment where accesses of multiple cores are intermingled.
- Since the proposed algorithm identifies the *dominant ways*, the number of these ways can also be known. Therefore, the *WADD* uses this scheme to dynamically match the number of *way candidates* appropriate to the current situation. Moreover, since this algorithm identifies the number of *dominant ways* for each core, it can quickly match the number of *way candidates* for each core even in a situation where the access patterns of multiple cores are intermingled.

The rest of the paper is organized as follows: Section II introduces the related research works. Our motivations are described in Section III, and the design of the *WADD* is proposed in Section IV. In Section V, experimental settings and results are discussed. Finally, Section VI concludes the paper.

II. RELATED WORKS

Existing studies on saving dynamic energy for set-associative caches apply *way prediction* [9]–[20] and pre-determined cache misses [21]–[25]. Existing *way prediction* schemes can be categorized into schemes that utilize the *recently-based locality* property [9]–[14], schemes that utilize the regularity of the cache access pattern [16]–[19], and a scheme that combines them [20]. Schemes for pre-determining cache misses use *partial tag comparison* [21]–[23] or the *modified Bloom filter* [24], [25].

The *most recently used* (*MRU*) scheme [11] considers a single recently accessed cache way (i.e., the MRU) as the *way candidate* for subsequent cache access. This scheme is quite simple and easy to implement while significantly reducing power consumption. This *MRU* scheme is one of the most popular approaches among the *recently-based locality* schemes. However, it exhibits low prediction accuracy in the L2 cache because of its vulnerability to high associativity and fragmented access patterns. However, since the existing *recently-based locality* scheme reflects program locality, it can be helpful even in a high-associativity L2 cache if the fragmentation problem can be managed. Therefore, other *way prediction* schemes have been applied to enhance the prediction accuracy by exploiting the advantage of the *recently-based locality*.

Way determination schemes [16]–[19] are helpful when the current access pattern is incompatible with the *recently-based locality* property or when the program has regular reference intervals. Such schemes determine the way to be accessed for the following cache reference from formalized access pattern analysis. Thus, they require additional memory to store

address information. This scheme can achieve significantly higher prediction accuracy and reduce power consumption if the working set size is smaller than the available cache size or if the data reuse interval is constant. However, performance is highly dependent on the size of the additional memory required to store the address information. Consequently, a significant amount of memory is required to achieve a high prediction accuracy. Additionally, if the current working set size is larger than the available cache size or the current working set has various data *re-reference intervals* (*RRI*s), the *way determination* schemes cannot ensure a sufficiently accurate *way prediction*.

Access mode prediction (*AMP*) [18] is a *way determination* scheme that adopts a *multicolumn-based way prediction* algorithm to improve prediction accuracy. The *multicolumn-based* algorithm updates the *major location* and the least significant $\log_2 n$ bits of the *major location*'s tag for an n-way set-associative cache in chronological order based on the *recently-based locality* property. When a cache reference occurs, it accesses the cache as a direct-mapped cache with *major location* information. The *AMP multicolumn-based way prediction* scheme exhibits high prediction accuracy even for a high-associativity cache because it takes advantage of the *recently-based locality* methods. However, it is only valid for applications requiring a limited amount of memory with short *RRI*s. Even with a short *RRI*, when a massive memory operation is executed, the *recently-based locality* is compromised because of fragmented access patterns. Furthermore, the *way prediction* accuracy is insufficient when an application runs with diverse *RRI* because of the thrashing issue of the *major location*. Since this condition usually occurs in high-end applications with shared lower-level caches, such algorithms are unsuitable for multi-core systems.

Recognizing a precise access pattern to ensure sufficient prediction accuracy in a shared L2 cache is not an easy task, given the diverse *RRI*s generated by fragmented patterns and intermingled accesses from multiple cores. Hence, the *Way Affinity Table and Look-Ahead Buffer* (*WAT+LAB*) [20] algorithm, applied in a multi-core environment, focuses on the *sequential access property*. The *WAT+LAB* algorithm manages the way number information according to the processor ID to alleviate the *sequential access property*. Access patterns are classified into two groups for block-level processing based on the *sequential access property* and the *recently-based locality* (i.e., the *way affinity property*). However, *WAT+LAB* has the premise that most shared cache accesses occur because the given working set is larger than the available cache size. Consequently, a substantial part of the shared cache access possesses the *sequential access property*. Thus, the *WAT+LAB* algorithm is suitable for large-scale memory operations, such as matrix and digital signal processing. However, such schemes are unsuitable for shared cache because intermingled access from multiple cores generates diverse *RRI*s. As a result, the *sequential access property* is compromised. Hence, these schemes are unsuitable for multi-core systems and cannot discriminate the access pattern characteristics correctly. Consequently, the *way prediction* accuracy deteriorates remarkably.

Since identifying the proper access pattern is crucial in improving the *way prediction* accuracy, the *dynamic per history length adjustment policy* (*DHL*) [15] adopts a history-based algorithm, which exhibits the advantage of accurately classifying each access pattern. This algorithm selects only the *valid history* for the current access pattern and utilizes it for *way prediction* in the L2 cache with a fragmented locality. Furthermore, according to this prediction result, the number of *way candidates* is adjusted to secure coverage in a high-associativity cache. However, the history of access patterns from multiple cores to the shared cache is mingled in a CMP environment. Thus, there is a limitation in that the *valid history* information is mixed and cannot be sufficiently secured.

In the *way halting* (*WH*) cache architecture [21], the proposed scheme can reduce the number of active ways by pre-determining a cache miss instead of using *way prediction*. *WH* applies a fully associative halt tag array that stores the least significant four tag bits of each way. This halt tag array performs comparisons with the four least significant tag bits of the address to detect non-selected ways in advance and reduce unnecessary accesses. Additionally, when halt tag hits do not exist, misses are detected in the decoding cycle, thus significantly reducing the cache miss penalty. However, the halt tag array cannot specify the current set because it compares the partial tag with each fully associative array per way. Thus, since the halt tag hit does not guarantee cache hits in the current set, the energy efficiency decreases because of false positives. Moreover, the lower-level cache has a relatively large number of sets and requires many comparisons, which increases the overhead of the fully associative halt tag arrays.

The *way-halted prediction* (*WHP*) [22] applies *way prediction* to the *WH* to improve energy efficiency in excessive halt tag hits caused by false positives of *WH*. If the number of halt tag hits exceeds one, the *WHP* designates a *way candidate* using the *MRU* algorithm. If the *way candidate* (determined by the *MRU* algorithm) is a halt tag miss, *WHP* accesses all the ways with a halt tag hit. This scheme can achieve more energy savings than *WH* when the *way prediction* accuracy is guaranteed. However, high prediction accuracy cannot be expected in a lower-level cache with high associativity because of the weak locality property of fragmented patterns. *WHP* also applies a fully associative halt tag array, which increases the overhead in the lower-level cache.

The *segmented tag cache* (*STC*) architecture [23] also proposes a scheme to avoid unnecessary data array access by detecting non-selected ways. The *STC* applies a modified tag array, which supports the following two access modes: the *partial access mode*, which first accesses a small number of low-order tag bits, and the *full access mode*, which accesses all the tag bits. During this time, the *partial access* time must be shorter than the data array decoding time. In this algorithm, the non-selected ways and the correct way are effectively

TABLE 1. Cache configuration for way access dominance analysis.

System	Single-core	Multi-core
Block Size	64 byte	
Replacement Policy	LRU	
L1-i	4-way 32 KB	
L1-d	8-way 32 KB	
L2	16-way 4 MB	16-way 4 MB shared

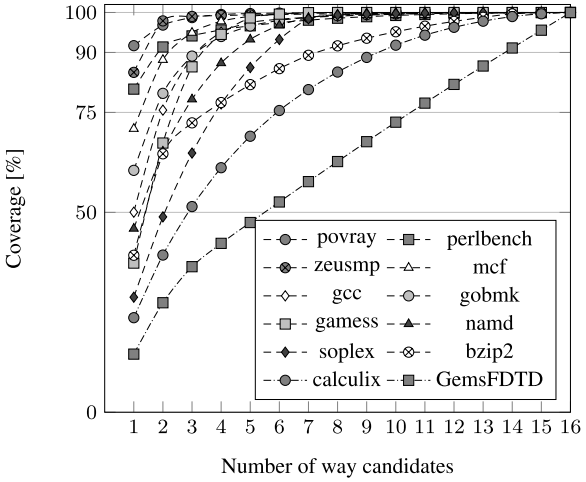

FIGURE 1. Way access coverage for each workload.

TABLE 2. Number of way candidates to secure each coverage (single-core).

Workload	> 50%	> 75%	> 90%
povray	1	1	1
perlbench	1	1	2
zeusmp	1	1	2
mcf	1	2	3
gcc	1	2	4
gobmk	1	2	4
gamess	2	3	4
namd	2	3	5
soplex	3	4	6
bzip2	2	4	8
calculix	3	6	10
GemsFDTD	6	11	14

distinguished. Therefore, this algorithm can achieve excellent performance in the L1 cache. However, unlike the L1 cache, in the L2 cache with high associativity, the *partial access* time exceeds the data array decoding time, thus increasing each access delay.

III. WAY ACCESS DOMINANCE DETECTION

We analyze each cache set's *way access dominance* for each access pattern to find a scheme to improve the *way prediction* accuracy in a shared L2 cache with high associativity. Accesses to the L2 cache have fragmented patterns because of the replacement policy of the L1 cache. Moreover, access patterns from multiple cores are intermingled in the shared cache, further fragmenting the access patterns. Since this characteristic weakens the access patterns' *recently-based locality* and makes the *RRI*s diverse, it is not suitable for existing *way prediction* schemes.

Therefore, we focus on implementing an energy-efficient *way prediction* algorithm, which does not impair performance significantly while reducing power consumption by selecting

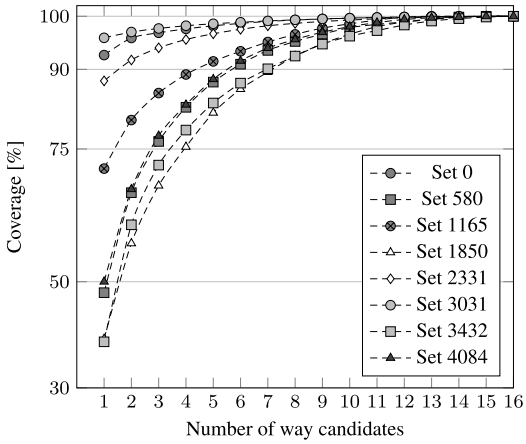

FIGURE 2. Way access coverage for each cache set for 'perlbench'.

TABLE 3. Multicore workload group configurations.

GRP1	tonto / gamess / dealII / povray
GRP2	namd / perlbench / gromacs / astar
GRP3	gromacs / omnetpp / GemsFDTD / gamess
GRP4	astar / gamess / sjeng / tonto
GRP5	povray / dealII / namd / omnetpp
GRP6	GemsFDTD / libquantum / gobmk / bzip2
GRP7	h264ref / namd / gobmk / gcc
GRP8	perlbench / h264ref / hmmer / gcc

an appropriate number of possible *way candidates*. For this purpose, each access pattern's trend of *way access dominance* is analyzed when several access patterns are intermingled in the L2 cache. We perform this analysis on each cache set. When two or more consecutive misses occur in the cache set, it is decided that the access pattern has changed. We consider the section between these consecutive misses and other consecutive misses as a *partial access pattern*. Table.1 shows the cache configuration used to analyze each workload's trend of *way access dominance* in the high-associativity L2 cache. We use the Sniper multi-core simulator [28] for this analysis.

Fig.1 shows the coverage according to the number of *way candidates* based on the number of ways accessed by each workload within the same *partial access pattern* on a single-core system. In the case of `povray`, just one *way candidate* can cover 91.7% of the total way access. On the other hand, `GemsFDTD` requires at least six *way candidates* to cover more than 50% of the total way access. Table.2 shows the number of *way candidates* required for the workloads in Fig.1 to secure a given coverage within a *partial access pattern*. Fig.1 and Table.2 show that several *way candidates* are required to achieve accurate *way prediction* in a high-associativity cache. Additionally, the required number of *way candidates* for each workload is different. Fig.2 shows the *way access dominance* shown on different cache sets during `perlbench`. Even for the same workload, it can be observed that each cache set should be controlled individually because the number of required *way candidates* is different for each cache set.

Fig.3 shows the coverage according to the number of *way candidates* based on the number of ways accessed by each workload group within the same *partial access pattern* on a quad-core system. The workload groups used in this

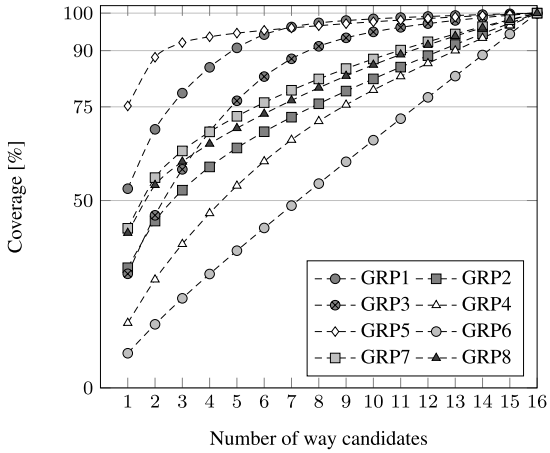

FIGURE 3. Way access coverage for each workload group.

TABLE 4. Number of way candidates to secure each coverage (Multi-core).

Workload Group	> 50%	> 75%	> 90%
GRP1	1	3	5
GRP2	3	8	13
GRP3	3	5	8
GRP4	5	9	13
GRP5	1	1	3
GRP6	8	12	15
GRP7	2	6	11
GRP8	2	7	12

process are configured by randomly selecting four workloads in the SPEC CPU2006 benchmark [26]. Table.3 shows the composition of each workload group. Even in a multi-core system, the *way access dominance* varies depending on the characteristics of the workloads that comprise each workload group. Table.4 shows that *way access dominance* in multi-core systems is relatively less visible than in single-core systems. Moreover, Table.5 shows that the multi-core system has different trends of *way access dominance* for each core. Because of this characteristic, *way prediction* schemes targeting shared caches require additional consideration of fragmented access patterns than schemes targeting private caches.

However, that advantage is only valid if the appropriate *way candidates* are selected. Fig.4 shows the percentage of ways actually accessed in the entire cache set during each workload in a single-core system. Except for workloads, such as `povray` and `solplex`, where accesses are dominant on specific ways, it can be observed that most of the access patterns access multiple ways at similar ratios. This result means that the number of ways used by one *partial access pattern* is limited, but the location of the way each *partial access pattern* uses changes continuously. Therefore, the *way prediction* scheme must correctly predict the location of each *way candidate* and the number of *way candidates* at that time.

Therefore, to achieve high-efficiency *way prediction* in the shared L2 cache, it is necessary to determine the location and number of *dominant ways* in the current *partial access pattern*. Moreover, it is necessary to properly adjust the number and location of the *way candidates* based on this

TABLE 5. Number of way candidates to secure each coverage.

Workload Group	Core	> 50%	> 75%	> 90%
GRP1	**0**	1	1	2
	1	2	3	5
	2	1	1	2
	3	1	1	1
GRP2	**0**	2	5	9
	1	2	6	11
	2	2	6	11
	3	3	8	13
GRP3	**0**	2	4	9
	1	1	2	5
	2	1	1	1
	3	2	4	5
GRP4	**0**	1	1	3
	1	2	4	6
	2	2	7	12
	3	8	12	15
GRP5	**0**	1	1	2
	1	1	1	1
	2	2	5	8
	3	1	2	5
GRP6	**0**	6	11	14
	1	8	12	15
	2	7	12	14
	3	7	12	14
GRP7	**0**	2	4	6
	1	2	5	8
	2	2	7	12
	3	1	2	8
GRP8	**0**	2	6	10
	1	2	4	7
	2	1	3	6
	3	1	2	7

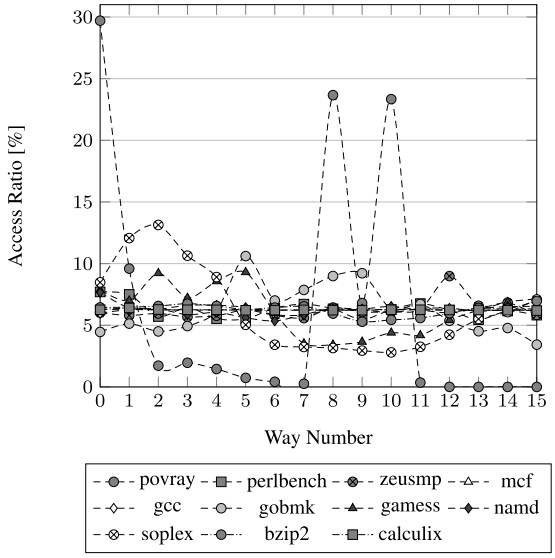

FIGURE 4. Way access ratio for each workload.

analysis. In a shared L2 cache, accesses from multiple cores are mixed, and the access ratio of each core changes. Hence, the changes in each *partial access pattern* also vary. To meet this need, we propose a *way prediction* algorithm that can detect *dominant ways* for each core and adjust the number and location of *way candidates* for each cache set in real-time.

IV. WAY-PREDICTION BASED ON WADD

The proposed *WADD* structure shown in Fig.5 is implemented based on the above requirements. It consists of ***way counter***s (which count the number of accesses to each way in the set), a ***selective way activator*** (which selects *way*

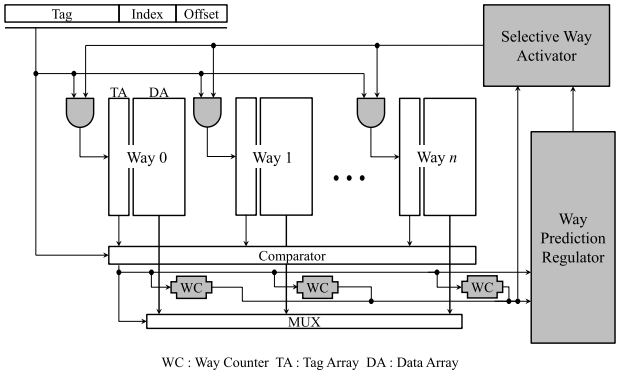

FIGURE 5. Proposed structure block diagram.

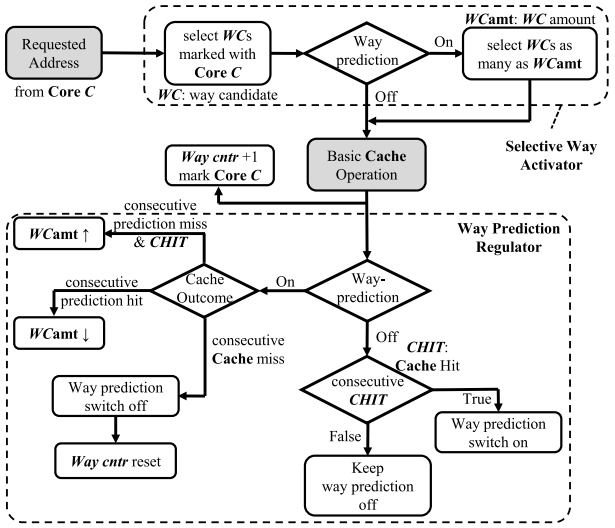

FIGURE 6. Flow chart of proposed algorithm.

candidates), and a ***way prediction regulator*** (which controls the number of *way candidates*). Since ***way counter***s continuously monitor the *way access dominance* for each core, they are the basis for the detection of *dominant ways*. The ***selective way activator*** selects the *way candidates* based on this basis. The ***way prediction regulator*** records the history of cache hit/miss outcomes and *way prediction* results in the corresponding cache set. Thus, it quickly identifies changes in *partial access pattern*s based on this history. This fast identification compensates for the impact of locality worsened by fragmented access patterns in lower-level caches such as L2 caches. Each of these components operates as follows.

The ***way counter*** consists of a 2-bit saturating counter that counts the access according to the cache outcome for the corresponding way in the set and a 4-bit marking bit (for a quad-core system) that indicates the core that has accessed the data stored in a corresponding way. These counters continuously monitor *way access dominance* by counting access to each way. This continuous monitoring is transmitted to the ***selective way activator***. In this way, the monitoring of each ***way counter*** continues while the *partial access pattern* is maintained. And then, when it is determined that the *partial access pattern* changed due to consecutive misses in the corresponding cache set, *WADD* resets the ***way counter***s correspond to the core. The overhead to implement this operation is estimated to be close to 250 transistors for each cache set because 2-bit saturating counter and 4-bit marking bits are allocated every way.

The ***selective way activator*** selects the *way candidates* among the ways marked with the currently accessed core. If the *way prediction* is turned on, the ***selective way activator*** selects as many *way candidates* as the ***way prediction regulator*** specifies. At this time, the ***selective way activator*** selects the *way candidates* in the order of the highest ***way counter*** value (*dominant ways*). If several ways have the same ***way counter*** value, the ***selective way activator*** determines the priority in the MRU order. Then, when the cache finds the correct way, the priority value of that way (determined by the ***selective way activator***) is sent to ***the way prediction regulator***. If the *way prediction* is turned off, the *WADD* determines the ways marked with a currently approached core as *way candidates*. Therefore, it is possible to continuously reduce the waste of dynamic energy regardless of whether the *way prediction* is turned on or not. For each cache set, the overhead to implement this operation is estimated to be more than 200 transistors because it allocates comparators to compare values received from way counters and AND gates to control access to the tag array.

The ***way prediction regulator*** helps in energy-efficient *way prediction* by regulating the number of *way candidates* or switching *way prediction*. The ***way prediction regulator*** stores the history of cache results, *way prediction* results, and priority values of the correct way for each core. It also records whether the difference between the number of *way candidates* determined by the ***way prediction regulator*** and the priority value of the correct way is zero. The history stored in this way becomes a criterion for determining the number of *way candidates*. Suppose continuous cache hits occur in one core while the *way prediction* is turned off. In this case, the ***way prediction regulator*** determines that a new *partial access pattern* has been detected and turns on *way prediction* for the corresponding core. The initial number of *way candidates* is determined as the number of ways whose ***way counter*** value is one or more among the ways marked with the corresponding core. With *way prediction* turned on, when successive prediction hits occur, the ***way prediction regulator*** attempts to reduce *way candidates* for efficient *way prediction*. Suppose it is confirmed through the history of the core that the difference between the number of *way candidates* and the priority value of the correct way is non-zero during successive prediction hits. In this case, the ***way prediction regulator*** decides to reduce the number of *way candidates*. On the other hand, if prediction misses occur continuously despite consecutive cache hits, the ***way prediction regulator*** attempts to increase prediction accuracy by designating the maximum value among the priority values of the most recent correct way stored in history as the following number of *way candidates*. Subsequently, if consecutive cache misses occur in one core, the ***way prediction regulator*** determines that the

partial access pattern has changed. Thus, the ***way prediction regulator*** prepares for a new *partial access pattern* by initializing the ***way counter*** values marked by the corresponding core while turning off the *way prediction* for that core. After that, when turning on the *way prediction* again, the number of *way candidates* for the same core is retrieved. For each cache set, the overhead to implement this operation is estimated to be less than 200 transistors because comparators are required to record and compare various histories for each core.

Changes in *partial access pattern*s can be quickly identified because *WADD* monitors the *way access dominance* using multiple counters and manages the various histories for each core. This quick identification and application make it possible to quickly respond to each access pattern in fragmented patterns caused by the L1 cache replacement policy and mixed access of multiple cores, making it possible to implement efficient *way prediction*. For each cache set, *WADD* requires approximately 650 transistors as overhead for all implementations. In a 16-way set-associative cache structure with a cache block size of 64-byte, the overhead of the proposed scheme is estimated to be close to 1% of the total L2 cache size. Preparing for the next *way prediction*, including the ***way prediction regulator***'s cache result update and ***way counter*** update, has no significant impact on latency, as it proceeds from the time the access to the L2 cache is completed until the subsequent access to the same set index occurs.

V. EXPERIMENTAL RESULT

A. METHODOLOGY

Table.3 in Section III presents the workload groups used to evaluate multiple algorithms in a shared cache environment. These groups are used with reference inputs in the Sniper multi-core simulator [28]. The cache configuration is a 64-byte block of a 16-way 4 MB shared L2 cache for a quad-core processor. This cache consists of a multi-bank cache with eight banks. Based on this configuration, the dynamic power dissipation and latency parameters are measured using CACTI [29] on the 32 nm process. Measured parameters consist of power and delays consumed by various components such as decoders and output drivers for each access. We put these parameters into the equations used in [22] to model the energy and delay.

In the case of energy, the fundamental energy consumption, including the decoder and output driver of the tag array or data array, is 93.16 pJ based on the conventional cache without *way prediction*. The overhead of each scheme is added to this value. In the case of the *MRU*, the overhead is negligibly small. The *WHP* has an overhead of 6.00 pJ due to the increased set index in the L2 cache. Attached is an overhead of 1.39 pJ for *STC*, 0.84 pJ for *DHL*, and 1.14 pJ for *WADD*. At this value, additional energy of 3.24 pJ (3.07 pJ in the case of *WHP*) is consumed per *way candidate*. When a *way prediction* miss occurs, 0.14 pJ of energy and the energy of other way access are consumed because the tag array is reaccessed. In addition, a penalty of 1.03 nJ is added in the case of a cache miss.

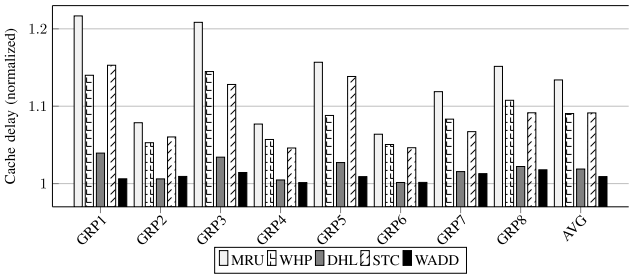

FIGURE 7. Cache access delay normalized to the conventional cache.

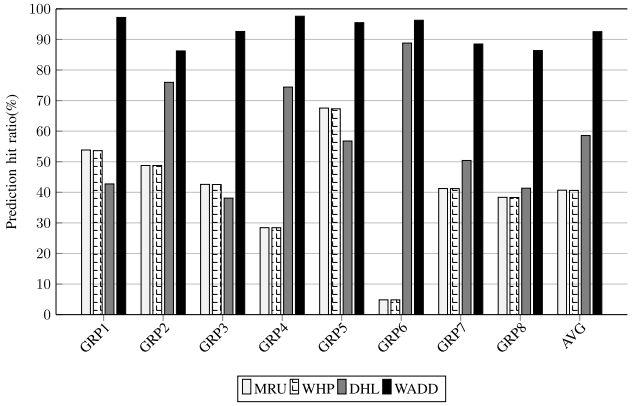

FIGURE 8. Way prediction hit ratio (%) for different algorithms.

In the case of delay, the latency is defined as 12 cycles when a cache hit occurs based on the conventional cache. The *STC* has an overhead of two cycles because it requires a lot of *partial access* time in the L2 cache environment. If a *way prediction* miss occurs, the tag array is reaccessed, and there is a penalty of four cycles. In addition, in the case of a cache miss, a penalty of 73 cycles is added.

B. CACHE ACCESS DELAY

Fig.7 shows the delay of each cache access for the *MRU*, *WHP*, *DHL*, *STC*, and *WADD*. The values are normalized to the access delay in a conventional cache without *way prediction*. In a shared L2 cache, the access patterns from multiple cores are intermingled, whereas the access patterns are fragmented in the L1 cache. This environment results in inferior accuracy of the *recency-based locality* scheme. Fig.8 shows that the *MRU* algorithm has a low accuracy of approximately 41%. The scheme using a single *way candidate* exhibits a limitation in performing *way prediction* based on the L2 cache's fragmented pattern with high associativity. We can check this limitation through the access delay.

In contrast, the *DHL* and *WADD* algorithms apply multiple *way candidates* to ensure prediction accuracy. Consequently, these algorithms achieve an accuracy of 1.4 and 2.3 times, respectively, more than the *MRU* algorithm. Using multiple *way candidates* leads to a higher dynamic energy consumption than using one *way candidate*. However, the access delay is reduced by reducing the prediction-miss penalty through high accuracy.

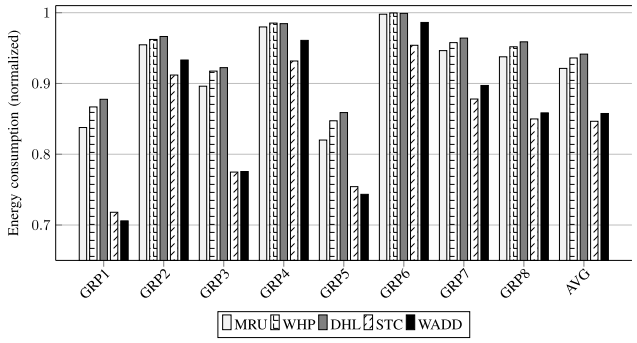

FIGURE 9. Energy consumption normalized to the conventional cache.

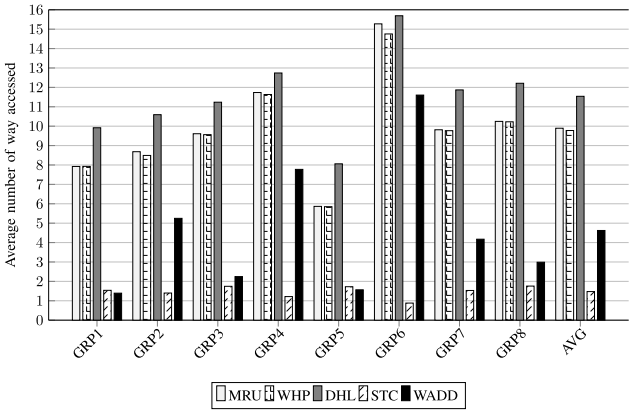

FIGURE 10. Average number of cache way accessed.

The *WHP* applies the *MRU* algorithm for *way prediction* and exhibits an accuracy similar to that of the *MRU* algorithm. The *WHP* pre-determines the non-selected ways using the halt tag before *way prediction*. Because of this process, the *WHP* can reduce unnecessary way accesses and determine cache misses quickly. However, the halt tag of *WHP* is not as accurate as the *partial access* of *STC*. The *WHP* false positives are particularly noticeable in L2 caches with a large number of cache sets. Consequently, the filtering of the halt tag cannot efficiently reduce the access delay.

On the other hand, as shown in Fig.7, *STC* exhibits the second-highest access delay. This result is related to the timing-conflict problem of the *STC*. In the L1 cache, which is the original target of the *STC*, the *partial access* time does not exceed the decoding time. Therefore, the *STC* does not have a delay overhead. However, an L2 cache with high capacity and associativity requires a high *partial access* time. For this reason, the *partial access* time significantly exceeds the decoding time, thus increasing the overall delay. Therefore, the *STC* exhibits the second-highest delay in this experiment, targeting the L2 cache.

C. DYNAMIC ENERGY CONSUMPTION

Fig.9 shows the dynamic energy consumption of each cache access for the *MRU*, *WHP*, *DHL*, *STC*, and *WADD*. The values are normalized to the energy consumption in the conventional cache without *way prediction*. In this experiment, the *WHP* and *DHL* exhibit average energy consumption without significant improvement, despite using a more complex structure than the *MRU* algorithm. The *WHP* finds non-selected ways in advance using fully associative halt tag arrays to reduce unnecessary way accesses. However, since the *WHP* allocates these halt tag arrays for each way, it cannot specify the current set. In other words, there is a false positive that the halt tag hit does not guarantee the cache hit in the current set. This problem deteriorates, especially in large-capacity L2 caches. For example, in the 16-way 4 MB shared L2 cache with the 64-byte block used in this experiment, the number of the set index is 4096. If a hit occurs in one entry among 4096 entries in the halt tag array, a halt tag hit occurs. Consequently, Fig.10 shows that the WHP cannot reduce the average number of ways accessed than the *MRU* algorithm. Additionally, each of the fully associative halt tag arrays must compare 4096 entries per cache access. Consequently, the *WHP* energy efficiency is further reduced because it requires more dynamic energy consumption in the L2 cache than in the L1 cache.

The *DHL* adopts a history-based algorithm to classify each access pattern accurately. This feature is required to increase the energy efficiency of L2 caches with fragmented access patterns. However, the access patterns from multiple cores are intermingled in the shared cache, causing further fragmentation. For this reason, there is a limit to how the *DHL* recognizes the access pattern and secures *valid history* information sufficiently. Additionally, once a cache miss occurs, the *DHL* returns to the *patterning reset* state. It takes at least three consecutive cache hits and two consecutive prediction hits to return to the *power-saving mode*. Thus, in a shared L2 cache where frequent cache misses occur due to frequent pattern changes, the *DHL power-saving mode* is maintained for a short period. The *DHL* performs *way prediction* only for 45.7% of the total cache hit access in this experiment.

The *STC* exhibits the lowest energy consumption on average. Since the *STC* pre-determines non-selected ways through the *partial access mode*, there are very few unnecessary way accesses, including sensing errors caused by *column-wise data randomization*. Hence, it exhibits relatively low energy consumption, including the energy overhead for *partial access*. Furthermore, even if there is no partially matched way in the *partial access*, cache misses can be immediately determined without unnecessary tag array access, thus further reducing unnecessary way accesses. Fig.10 shows that the average number of ways accessed by *STC* in a 16-way set-associative cache is less than two.

The second algorithm exhibiting the lowest energy consumption is the *WADD* proposed in this paper. The *WADD* detects the *dominant ways* quickly and responds appropriately to increase energy efficiency. Although this algorithm does not have a small number of way accesses because of multiple *way candidates*, the actual dynamic energy consumption differs from the *STC* by only approximately 1.1% because of the increased prediction accuracy. The *WADD* uses fewer ways than other *way prediction* algorithms such as *MRU*, *WHP*, and *DHL* while still exhibiting considerable

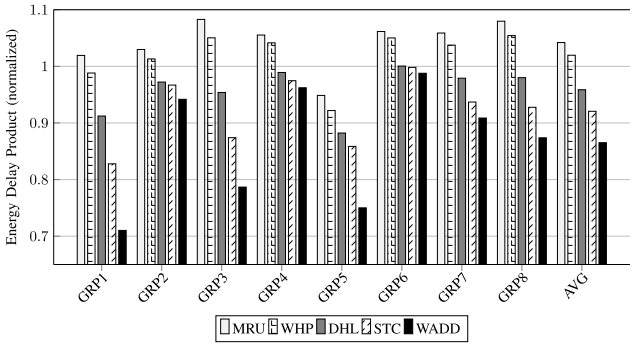

FIGURE 11. Energy-delay product normalized to the conventional cache.

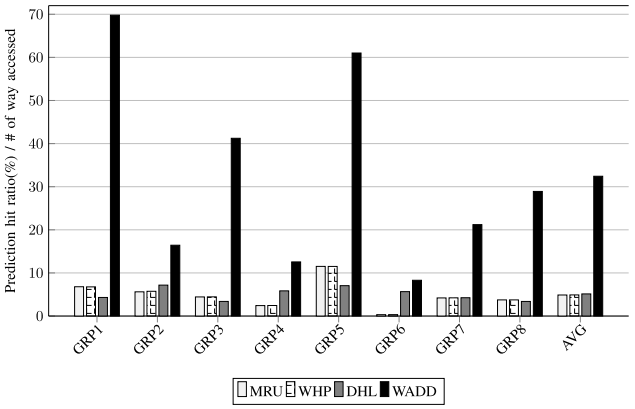

FIGURE 12. Way prediction hit ratio divided by the number of way accessed.

prediction accuracy. Additionally, even in a shared L2 cache, where *partial access pattern* changes are frequent because of quick pattern adaptation, *way prediction* can be performed on an average of 92.3% of the total cache hit accesses. Thus, the *WADD* can increase the energy efficiency while inducing less *way prediction* penalty.

D. ENERGY EFFICIENCY

Fig.11 shows an Energy-Delay Product (EDP) graph. The halt tag of the *WHP* pre-determines non-selected ways, but several false positives occur in the lower-level cache with a large capacity. Thus, the halt tag cannot effectively filter out unnecessary way accesses. Fig.10 shows that the *MRU* and *WHP* algorithms, which use the same *way prediction* scheme, have a similar average number of ways accessed. Thus, the *WHP* can slightly reduce the delay compared to the *MRU* algorithm. However, this cannot reduce the dynamic energy because of the overhead of operating the halt tag. Consequently, the *WHP* does not exhibit significant energy efficiency improvement compared to the *MRU* algorithm.

Since the *DHL* uses multiple *way candidates*, it consumes slightly more dynamic energy than the *MRU* algorithm. However, *DHL* improves the *way prediction* accuracy by 1.4 times. Consequently, the *DHL* can improve the overall energy efficiency by reducing the delays by approximately 6% compared with the *MRU* algorithm. However, the *DHL* has 45.7% of the total cache hit accesses that the *way prediction* has operated and exhibits a prediction accuracy of 58.6%. In other words, the *DHL* can further improve energy efficiency by improving its *way prediction* scheme.

The *STC* pre-determines non-selected ways by applying the *partial access mode*, thus filtering out the most unnecessary way accesses. Consequently, the *STC* consumes nearly 8% less dynamic energy than the *MRU* algorithm, including the energy overhead for *partial access*. However, in the *partial access mode*, false positives can also occur. For a cache hit, the *STC* uses an average of approximately 1.8 ways per access. Moreover, even if the *partial access mode* fails to pre-determine the cache miss, the *STC* accesses about 1.2, which is the partially matched ways. This case accounts for approximately 56.9% of all cache misses. Therefore, the *STC* cannot significantly reduce the dynamic energy consumption and delay. Additionally, the *partial access* time becomes an overhead in the lower-level cache with a large capacity and high associativity.

The *WADD* uses counter-based *way access dominance detection* to perform efficient *way prediction* on the shared L2 cache with a mixture of fragmented access patterns. Consequently, the *WADD* can respond quickly to frequent pattern changes, and thus, it uses fewer than half the number of ways compared with the *MRU*, *WHP*, and *DHL*, exhibiting a 92.5% accuracy. Fig.12 shows the *way prediction* hit ratio divided by the number of ways accessed. We can observe that the *WADD* performs efficient *way prediction* by selecting appropriate *way candidates* utilizing the *way access dominance detection*. Consequently, since the *WADD* uses multiple *way candidates*, the dynamic energy consumed is approximately 2% more than the *STC*. However, the *WADD* exhibits the highest energy efficiency among the comparison groups by reducing the penalty through high prediction accuracy.

VI. CONCLUSION

This paper proposes a *way prediction* algorithm based on the *way access dominance detection* for a shared L2 cache with high associativity. Since the proposed algorithm continuously detects the *dominant ways*, it is possible to quickly identify the *partial access pattern* change & the trend of *way access dominance* and implement efficient *way prediction* in response to that. The additional overhead to support the *WADD* operation is considered in two ways: delay and energy. *WADD* begins *way access dominance detection* when the current L2 cache access operation is complete and prepares for the following *way prediction* before the subsequent L2 cache access occurs. Therefore, there is very little delay overhead. In contrast, the energy overhead for the operation of each ***way counter*** and ***way prediction regulator*** requires approximately 1% of the target conventional cache.

The disadvantage of *WADD* is that it cannot save dynamic energy reliably because it uses multiple *way candidates* to secure the *way prediction* accuracy. However, although it maintains high prediction accuracy, the number of ways accessed is less than half that of other *way prediction* algorithms, resulting in higher energy efficiency. Nevertheless,

the *WADD* still has room for further development. The percentage of accesses with *way prediction* activated is 92.3% of the total cache hit accesses. This result is also related to *WADD* considering successive cache misses as the criterion for *partial access pattern* change and successive cache hits as the criterion for *way prediction* activation. This characteristic is the cause of the weakness that *WADD* cannot effectively respond when the toggle between cache hits and cache misses is repeated. This situation occurs at approximately 6% of the workload accesses used in the experiment. The results showed that the *WADD* exhibits a 96.7% *way prediction* activation rate, excluding this weakness. Therefore, if the response to this weakness can be improved, and this improvement can enhance the prediction accuracy, the *WADD* energy efficiency can be further improved.

REFERENCES

[1] A. Cristal, O. J. Santana, F. Cazorla, M. Galluzzi, T. Ramirez, M. Pericas, and M. Valero, "Kilo-instruction processors: Overcoming the memory wall," *IEEE Micro*, vol. 25, no. 3, pp. 48–57, May 2005.

[2] T. Wada, S. Rajan, and S. A. Przybylski, "An analytical access time model for on-chip cache memories," *IEEE J. Solid-State Circuits*, vol. 27, no. 8, pp. 1147–1156, Aug. 1992.

[3] J. Friedrich, R. Puri, U. Brandt, and M. Buehler, "Design methodology for the IBM POWER7 microprocessor," *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 1–14, May 2011.

[4] H. H. Najaf-abadi, N. K. Choudhary, and E. Rotenberg, "Core-selectability in chip multiprocessors," in *Proc. 18th Int. Conf. Parallel Archit. Compilation Techn.*, Raleigh, NC, USA, Sep. 2009, pp. 113–122.

[5] A. Sodani and C. Processor, "Race to exascale: Opportunities and challenges," in *Proc. IEEE/ACM 44th Ann. Int. Symp. Microarchitecture*, Jan. 2011, pp. 1–28.

[6] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "CPU DB: Recording microprocessor history," *Commun. ACM*, vol. 55, no. 4, pp. 55–63, 2012.

[7] C. Gilberto and M. Margaret, "Power prediction for Intel Xscale processors using performance monitoring unit events," in *Proc. Int. Symp. Low Power Electron. Design*, San Diego, CA, USA, 2005, pp. 221–226.

[8] B. Burgess, B. Cohen, M. Denman, J. Dundas, D. Kaplan, and J. Rupley, "Bobcat: AMD's low-power X86 processor," *IEEE Micro*, vol. 31, no. 2, pp. 16–25, Mar. 2011.

[9] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proc. 2nd Int. Symp. High-Perform. Comput. Archit.*, San Jose, CA, USA, 1996, pp. 244–253.

[10] M. Calagos and Y. Chu, "Hybrid scheme for low-power set associative caches," *Electron. Lett.*, vol. 48, no. 14, pp. 819–821, Jul. 2012.

[11] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proc. Int. Symp. Low Power Electron. Design*, New York, NY, USA, 1999, pp. 273–275.

[12] M. Zhang and K. Asanovic, "Highly-associative caches for low-power processors," in *Proc. 33rd Int. Symp. Microarchitecture*, Monterey, CA, USA, 2000, pp. 1–6.

[13] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proc. 34th ACM/IEEE Int. Symp. Microarchitecture*, Austin, TX, USA, Jan. 2001, pp. 54–65.

[14] B. Batson and T. N. Vijaykumar, "Reactive-associative caches," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Barcelona, Spain, 2001, pp. 49–60.

[15] H. W. Joo and E. Y. Chung, "DHL-cache: Dynamic per history length adjustment for low-power L2 cache," *Electron. Lett.*, vol. 52, no. 15, pp. 1297–1298, Jul. 2016.

[16] A. Sembrant, E. Hagersten, and D. Black-Shaffer, "TLC: A tag-less cache for reducing dynamic first level cache energy," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Davis, CA, USA, Jan. 2013, pp. 49–61.

[17] R. Min, W.-B. Jone, and Y. Hu, "Location cache: A low-power L2 cache system," in *Proc. Int. Symp. Low power Electron. Design*, Newport Beach, CA, USA, 2004, pp. 120–125.

[18] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design," *IEEE Micro*, vol. 22, no. 2, pp. 58–71, Mar. 2002.

[19] J. Dai and L. Wang, "An energy-efficient L2 cache architecture using way tag information under write-through policy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 102–112, Jan. 2013.

[20] C.-M. Chung and J. Kim, "Low-power L2 cache design for multi-core processors," *Electron. Lett.*, vol. 46, no. 9, pp. 618–620, 2010.

[21] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, pp. 34–54, Mar. 2005.

[22] N. B. Mallya, G. Patil, and B. Raveendran, "Way halted prediction cache: An energy efficient cache architecture for embedded processors," in *Proc. 28th Int. Conf. VLSI Design*, Bangalore, India, Jan. 2015, pp. 65–70.

[23] M. Kim, I.-J. Chang, and H.-J. Lee, "Segmented tag cache: A novel cache organization for reducing dynamic read energy," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1546–1552, Oct. 2019.

[24] G. Keramidas, P. Xekalakis, and S. Kaxiras, "Applying decay to reduce dynamic power in set-associative caches," in *Proc. Int. Conf. High-Perform. Embedded Architectures Compil.*, Ghent, Belgium, 2007, pp. 38–53.

[25] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.-H.-S. Lee, "Way guard: A segmented counting Bloom filter approach to reducing energy for set-associative caches," in *Proc. 14th ACM/IEEE Int. Symp. Low Power Electron. Design*, San Fancisco, CA, USA, 2009, pp. 165–170.

[26] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[27] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, New York, NY, USA, 2008, pp. 72–81.

[28] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, New York, NY, USA, 2011, pp. 1–12.

[29] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, Dec. 2011, pp. 694–701.

YUN-SEOK OH (Student Member, IEEE) received the B.S. degree from Yonsei University, Seoul, South Korea, in 2011, where he is currently pursuing the Ph.D. degree in electrical and electronic engineering.

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

EUI-YOUNG CHUNG (Member, IEEE) 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 the 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.