Regular Paper
Journal of Computing Science and Engineering,
Vol. 15, No. 3, September 2021, pp. 107-114

Cascaded Cache Based on Recently Used Order for Latency Optimization for IoT

Juhee Choi
Department of Smart Information Communication Engineering, Sangmyung University, Cheonan, Korea
jhplus@smu.ac.kr

Heemin Park*
Department of Software, Sangmyung University, Cheonan, Korea
heemin@smu.ac.kr

Abstract
The load-to-use latency of the L1 cache is one of the main factors to determine the performance of low power processors. As the clock frequency competitions become severe even in Internet of Things (IoT) devices, the access cycle to the L1 cache was increased to meet the timing constraints. Because previous studies assumed that every access to the data cache has the same latency, the latencies tend to be longer for higher performance. We propose a latency optimization method that uses a cascaded cache based on a recently used order that has several small banks with various latencies instead of one whole data cache with constant latency. In our cache, each bank contains the cache block according to its recently used order. The experiments show that the performance of our proposal is improved by 23.0%, with a 1.3% reduction in the dynamic energy consumption on average.

Category: Information Retrieval / Web

Keywords: Latency optimization; L1 data cache; Cache replacement policy

I. INTRODUCTION

A hierarchical cache system is adopted to alleviate the memory wall problem caused by the difference between the access time of the core and the main memory [1]. During memory access, the core tries to fetch instructions or data from level 1 (L1) cache instead of higher-level caches or main memory. If a cache hit occurs in the L1 cache, the core starts to execute with the cache block. The latency from the start of cache access to the forwarding of the cache block to the core is important for the L1 cache because it directly affects performance due to the in-order pipeline and low-complexity superscalar architecture. However, many researchers proposed to save the cache power consumption, such as way-prediction or filter cache [2-5], rather than optimizing the latency, because they assumed that the latency was fixed.

For modern computing environments, the load-to-use cycle of the L1 data cache is more than two cycles, even in the cores embedded in Internet of Things (IoT) devices [6]. As the pipeline is deeper and the performance competition becomes more intense in the market, the access latency of the cache is prolonged to meet the requirement of high clock frequency. Even though the

Open Access http://dx.doi.org/10.5626/JCSE.2021.15.3.107 http://jcse.kiise.org

Received 15 June 2021; **Accepted** 28 August 2021
*Corresponding Author

pISSN: 1976-4677 eISSN: 2093-8020

overall performance was improved by this approach, it still needs to be studied to mitigate the drawback of the multicycle latency of the L1 cache. Existing studies considering variable access latency for cache have been investigated [7, 8], but they have mainly focused on the L3 cache or last level cache (LLC).

To reduce the latency, we propose a novel cache architecture that consists of small banks to store cache blocks according to their recently used (RU) order. The latencies of the banks vary from a single cycle to several cycles. The most recently used (MRU) cache blocks are in the bank with single cycle latency, while the least recently used (LRU) cache blocks are in the bank with the longest cycle latency. Since MRU blocks are most frequently utilized, a substantial amount of cache access may be completed in a single cycle, resulting in a reduction in average latency. The simulation results show a significant performance improvement with a small power consumption overhead for keeping the RU order.

The rest of this paper is organized as follows. In Section II, related studies are summarized. The motivation of our proposal is described in Section III. Section IV shows a detailed description of the cascaded cache based on recently used order. The experimental setup and the results are given in Section V. Finally, we conclude this paper in Section VI.

II. RELATED WORK

Way-predictors have been studied to mitigate the performance degradation of the sequential access cache [2, 3]. Each set in a set-associative cache consists of several cache blocks. To reduce the unnecessary dynamic power consumption of the cache access, the data array in the cache was not accessed until the tag matching was done. As a result, sequential access to the tag array and data array increased data access latency as compared to parallel access cache, which searched the tag array and data array simultaneously. To overcome this drawback, one of the blocks was speculatively accessed in parallel with the tag matching by the way predictor. If the prediction was correct, it compensated the delay caused by the sequential access.

Filter cache was proposed as an approach to reduce the dynamic energy consumption [4, 5]. A small storage, known as the filter cache or L0 cache, was inserted between the core and the L1 cache. When the core requested instructions or data, it searched only the filter cache rather than the L1 cache. When the cache block is in the filter cache, it saves the energy consumption of the L1 cache access, while the latency for the L1 cache is prolonged. The filter cache is usually adopted by high performance cores, which is called micro-op cache for the L1 instruction cache only [9]. In practice, this means that the filter cache is not ideal for the small cores and the L1 data cache.

Non-uniform cache architecture (NUCA) was proposed to optimize the latency for an LLC, which consists of many slices [7, 8]. Since the LLC occupies a large portion of the die area, the slices are physically distributed throughout the chip. Therefore, the latencies of the slices naturally vary depending on their physical distance between the core and the requested cache block. To take advantage of its characteristics, the researchers for NUCA have studied the placement location, migration, and replication policy for the LLC. Even though the NUCA also adopts the concept of variable access latencies for slices, it utilized unique characteristics of the LLC such as multi-core environments, so that their works are not applied for the L1 cache.

Therefore, this paper suggests a novel method for optimization of the latency of the L1 cache by utilizing the behavior of the cache hits in IoT systems.

III. MOTIVATION

Our proposal is supported by the observation that the average latency of the L1 cache is related with the pattern of cache hit counts. To investigate the relationship, we built an analytic model for the average latency in terms of latencies of cache hits and the miss rate. Next, it was found that a large portion of cache hits occur in the MRU blocks by analyzing the cache hit pattern. Finally, we inferred that by placing the MRU blocks into the storage which has the lowest latency reduces the average latency of the cache by applying the result of analysis to the analytic model.

A. Analytic Model for Average Latency

Our idea started from revisiting the relationship between the average latency of the L1 cache access, the cache block hit counts, and the miss rate. To reveal the relationship in detail, we built an analytic model for the average latency of the L1 cache. Traditionally, the average latency of the L1 cache (T_{L1_avg}) with the fixed latency is defined as follows:

$$T_{L1\_avg} = \frac{H}{\mathrm{N}} T_{L1\_Hit} + \frac{(N-H)}{\mathrm{N}} T_{L2\_avg} \quad (1)$$

where N is the amount of memory accesses, H denotes the total L1 hit counts, T_{L1_Hit} indicates the fixed latency of the L1 cache hit, and T_{L2_avg} denotes the average latency of the L2 cache.

Next, we assumed that the access time varied among cache blocks in a set and T_n meant the latency was n cycles. If a cache block is loaded in n cycles, the total number of the cache hits is defined as H_n. Therefore, H is the summation of H_n.

$$H = H_1 + H_2 + H_3 + \cdots = \sum_{i=1}^{n} H_i \tag{2}$$

Next, we introduced the new average latency T':

$$T'_{L1\_avg} = \frac{H_1 H}{H\,N} T_1 + \frac{H_2 H}{H\,N} T_2 + \cdots + \frac{(N-H)}{\mathrm{N}} T_{L2\_avg}$$
$$= \sum_{i=1}^{n} \left(\frac{H_i}{\mathrm{H}} T_i \frac{H}{\mathrm{N}} \right) + \frac{(N-H)}{\mathrm{N}} T_{L2\_avg} \tag{3}$$

In addition, the L1 miss rate is usually defined as the portion of the cache misses over the total cache accesses. Thus, we obtained the following equation in terms of the L1 miss rate (M_{L1}).

$$T'_{L1\_Avr} = \sum_{i=1}^{n} \frac{H_i}{\mathrm{H}} T_i (1 - M_{L1}) + M_{L1} T_{L2\_Avr} \tag{4}$$

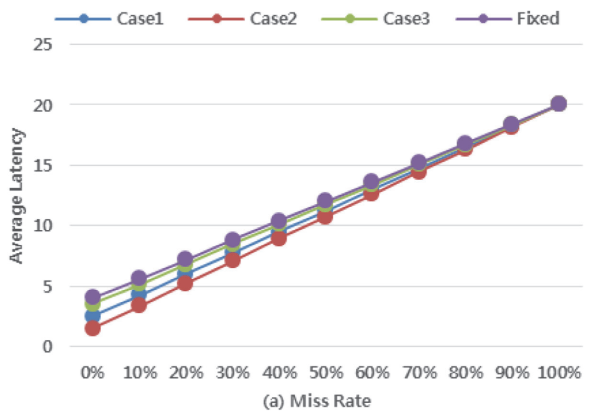
The new equation showed that the average latencies were determined by the L1 miss rate and the portion of H_n.

Fig. 1 shows the average latencies calculated by the Eq. (4). The lines with the solid circle represent the average latencies and the lower is better in Fig. 1(a). The Case/Fixed ratio indicated how much the average latency was decreased in Fig. 1(b); thus, a higher value was better. Case 1 denoted that the L1 hits were evenly distributed for each bank. For Case 2, 3/4 of the L1 hits occurred in MRU blocks and the other hits occurred equally, while 3/4 of the L1 hits occurred in LRU blocks for Case 3. The L2 average latency was assumed to be 20 cycles, while the hit latency cycle of the conventional L1 cache was four. As the miss rate increases, the Case/L1 ratios converged to 1, which showed that lower miss rate led to higher performance. In addition, when the hit counts of the MRU blocks occupied the higher portion of the total hits, the average latency was more reduced.

B. Behavior of Cache Hits

Recently, the low power processors for IoT devices gradually employed complex L1 caches which have multicycle-hit latency and set-associativity based on LRU replacement policy instead of single-latency direct-mapped policy. To gain insight into the advanced cache, we analyzed the behavior of the cache hits by finding the hit counts of each RU order.

Before proceeding with the discussion, we need to clarify the difference between the concepts of the "way" and RU order. When a cache block was inserted, the



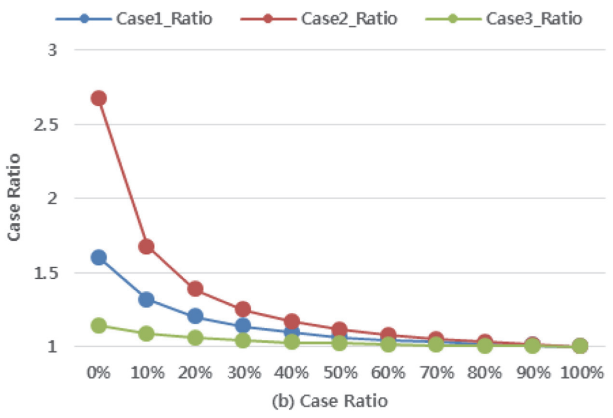


Fig. 1. Average latency for the variable access latencies: (a) miss rate and (b) case ratio. For Case 1, $H_1 = H_2 = H_3 = H_4 = \frac{H}{4}$. For Case 2, $H_1 = 3\frac{H}{4}$, $H_2 = H_3 = H_4 = \frac{1}{3}\frac{H}{4}$. For Case 3, $H_1 = H_2 = H_3 = \frac{1}{3}\frac{H}{4}$, $H_4 = 3\frac{H}{4}$.

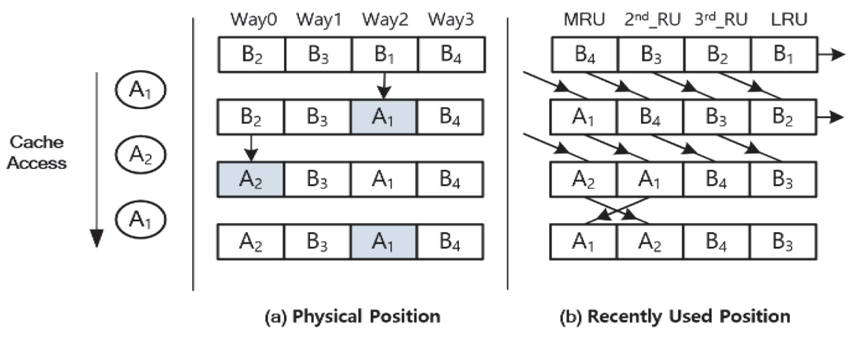


Fig. 2. (a) Physical position and (b) recently used position.

physical position of the "way" was fixed, while the RU order was changed if the other block was accessed. To help understanding the difference between the two concepts, we provided an illustration in Fig. 2. The cache in this example is a 4-way set-associative cache and four blocks were accessed as the sequence of B_1, B_2, B_3, and B_4. There was no order among the cache blocks for the physical position, so that the order of the "way" was not related to the cache blocks. However, the RU positions were determined by the time when the blocks were used. B_4 was located in the MRU position because it was the latest accessed block, while B_1 occupied the LRU position. It was assumed that the core tried to fetch A_1, A_2, and A_1. For the first access, A_1 replaced B_1 and A_1 became the MRU block. The RU positions of the rest of the blocks were changed, while their physical positions remain unchanged. Since the cache miss occurred for A_2, B_2 was evicted and A_2 was inserted. The next A_1 access modified the RU positions of A_1 and A_2, however, the physical position of A_1 remain unchanged.

Fig. 3 shows the results of three kinds of cache configurations which are 8 kB, 16 kB, and 32 kB 4-way set-associative caches because these configurations were mainly used for IoT devices [6]. The detailed simulation environments are described in Section V. The MRU and the LRU signified that the cache hits occurred in the most recently used blocks and the least recently used blocks. The second and third represent the portion of cache hits for second recently used blocks and third recently used blocks, respectively.

More than 82%, 88%, and 92% of cache hits occurred in the MRU blocks for the 8 kB, 16 kB, and 32 kB cache, respectively. The second recently used blocks were accessed as cache hits at 10.5%, 7.1%, and 5.8%.

Only 2.0%, 1.1% and 0.5% of cache accesses served in the LRU blocks. For only *lesie3d* and *hmmer*, more than 20% of cache hits occurred in second and third. In addition, as the cache size became larger, the portion of the MRU blocks increased. From the analysis of the access patterns, we found out that cache hits were concentrated on the MRU position.

This observation inspired us that reducing the cache hit latency for the MRU blocks led to decrement in the average latency. Assuming that H_n was the hit counts of each RU position, the Eq. (4) was rewritten as follows:

$$T'_{L1\_Avr} = \sum_{i=1}^{n} \frac{RU_i}{\mathrm{H}} T_i(1-M_{L1}) + M_{L1} T_{L2Avr} \qquad (5)$$

where RU_i indicates the hit counts of the ith RU order and n is the number of associativity. Since the equation is proportional to the portion of hit counts for T_1, it is advantageous to maximize the value of RU_1/H.

From the discussion above, we concluded that reducing the latency of the MRU blocks results in a decrease in average latencies. To implement this idea, a novel cache architecture was proposed.

IV. CASCADED CACHE BASED ON RECENTLY USED ORDER

We devised a new architecture known as a cascaded cache based on recently used order (CCRU). In the CCRU, the tag array and data array were split by the RU banks. The access cycle to each bank was decided by the order of the bank. To keep the RU order of the bank, the cache blocks in the same set, containing the requested block, were moved to the L1 cache on every access if necessary.

A. Overall Architecture

The CCRU consisted of small banks which have their own latencies instead of the same latency for all banks. The tag array and data array for each way were physically separated in the CCRU, as shown in Fig. 4, while the conventional cache consists of one tag array and one data array for all sets. The cache blocks of the CCRU were located in the banks corresponding to the RU order. The first bank or bank zero has a single-cycle latency. If the blocks in the fourth bank were accessed, four cycles

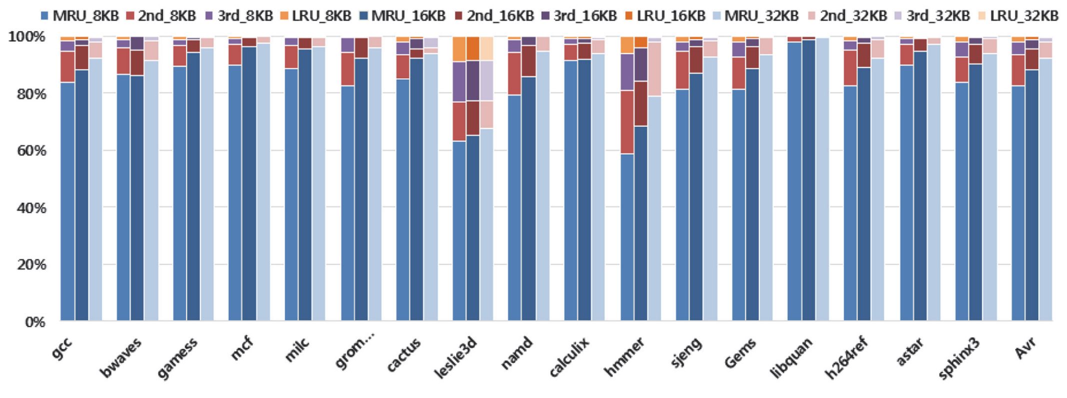


Fig. 3. Distribution of recently used order of cache hits. MRU represents the cache hits that occurred in the most recently used blocks.

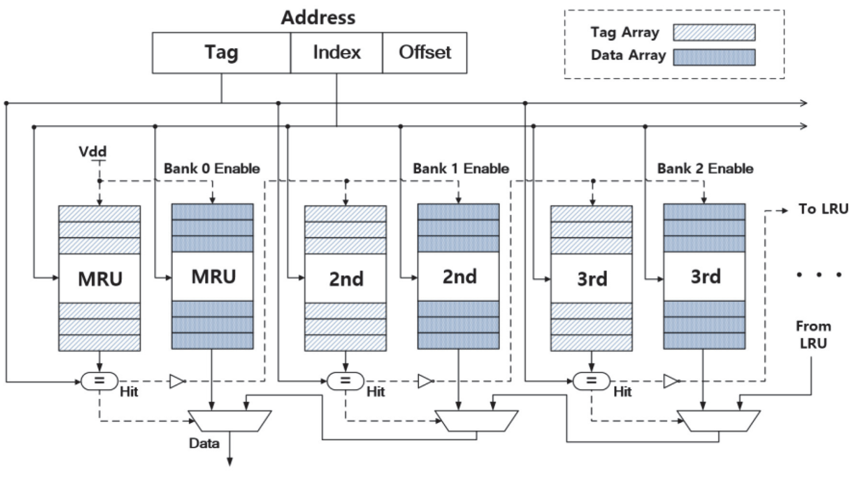


Fig. 4. Cascaded cache based on recently used order. The detailed logic for moving blocks among ways is omitted.

would be needed to fetch the block.

B. Cache Hit Sequence

When the core tried to fetch a cache block, only the first bank was searched. If the cache block was in that bank, the L1 cache can provide the requested block in a single cycle. However, if the block was not in the first bank, the next bank was enabled and searched. After tag matching for the second bank, it was determined whether the third bank needed to be accessed or not. Note that the access cycle of each bank itself was a single cycle. However, the total latency for requesting core was accumulated by each bank access. For example, four cycles were needed to fetch the cache block in the fourth bank: three cycles were used for the cache misses of the three banks and one cycle was for the hit latency of the fourth bank.

C. Bank Update Policy

When a cache block was inserted into a conventional cache, its contents were placed into the tag array and the data array, where they remain until eviction. However, in the CCRU, when a cache block was accessed, the tag information and the data of the block were moved to the MRU banks and the other blocks in the same set were moved to the corresponding RU banks. For example, when the cache block in the third bank was accessed, the block was forwarded to the MRU bank. The block which was in the MRU bank was moved to the second bank and the block in the second bank was shifted to the third bank. During this rearrangement of blocks, the extra dynamic energy was consumed compared with the conventional LRU cache. Therefore, keeping the RU order for each bank requires energy overhead.

V. EXPERIMENTAL RESULTS

A. Simulation Environments

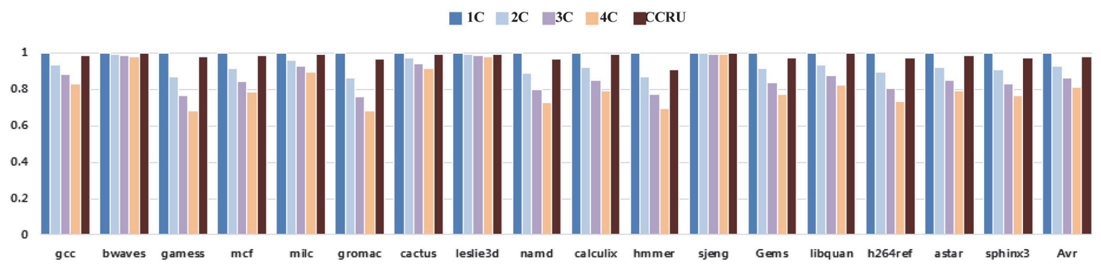
We simulated our study with the SPEC2006 benchmark suite [10]. The gem5 simulator was used to evaluate the performance and the energy consumption of our proposal [11]. In addition, one billion instructions were executed for each application with fast-forwarding one billion instructions to reduce the experiment time. Detailed information of the environment is shown in Table 1. To estimate the energy consumption and latency of the caches, the CACTI 7 cache model is applied with the technology of 0.22 nm [12].

B. Normalized Performance, Average Latencies, and Miss Rate

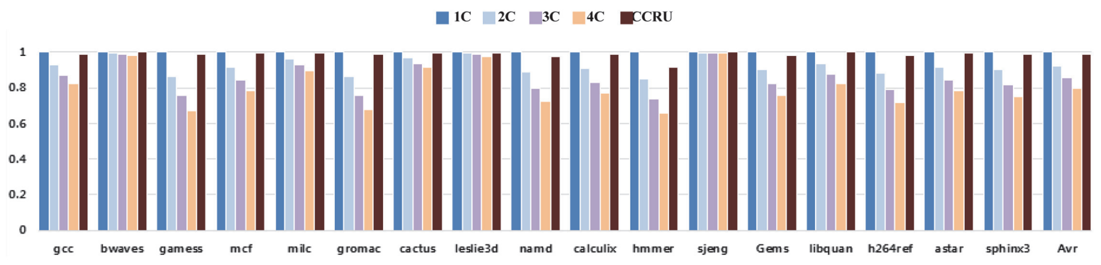
Fig. 5 provides the normalized performance in terms of IPC for our proposal and various latencies with L1 data

Table 1. Parameters of the simulation environments

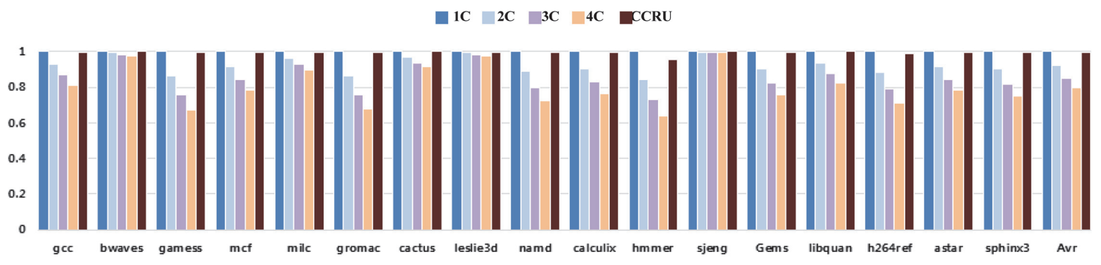
Parameter	Specification
L1 Instruction/Data cache	8/16/32 kB, 4-way set-associative, 32B line, access latency (1–4 cycles)
L2 unified cache	128 kB unified, 8-way, 32B line, access latency (20 cycles)
Memory	DDR SDARM, data bus width (64 bits), access latency (200 cycles)



(a)



(b)



(c)

Fig. 5. (a–c) performance with 8 kB, 16 kB, and 32 kB cache. 1C means the hit latency is a single cycle with the traditional cache. 2C, 3C, and 4C indicate that the hit latencies are two, three, and four cycles, respectively. The baseline of the normalization is 1C. 1C implies the upper limit of the performance because it always takes a single cycle for cache access.

cache. 1C, 2C, 3C, and 4C in Fig. 5 indicated that the fixed load-to-use latencies were one cycle, two cycles, three cycles, and four cycles with the conventional data cache. The standard of the normalization was the result of 1C because it implied the optimal performance. The IPC values of the 8KB cache, as shown in Fig. 5(a), were on average lower than 7.5%, 13.8%, and 19.1% with 2C, 3C, and 4C compared to 1C, respectively. The normalized values with 2C, 3C, and 4C were 0.921, 0.855, and 0.800 for the 16 kB and 0.920, 0.852, and 0.796 for the 32 kB as provided in Fig. 5(b) and 5(c).

The first thing found in Fig. 5 was that the trends of the performance improvement were similar regardless of the cache size. Overall, the CCRU achieved a 21.1%, 23.3%, and 24.6% speed-up compared to the 4C on average with 8 kB, 16 kB, and 32 kB cache, respectively. Thus, this indicates that our proposal is generally more efficient than the special configuration of the cache. In addition, the differences between 1C and our proposal were only 2.0%, 1.3%, and 0.7% for each cache size. As a result, the CCRU nearly reaches its maximum performance.

In general, the speed-up for the CCRU is proportional to the portion of the MRU hits over the total cache hits as shown in Fig. 3. However, for some programs such as *bwaves* and *leslie3d*, the performance improvements were only 1.8% and 1.9% unlike the other programs which have the nearly same portion of the MRU hits. To investigate this phenomenon, we showed the average latencies and the L1 cache miss rate in Fig. 6. Overall, the miss rates gradually decreased from 7.9% to 4.0% as the cache size increased from 8 kB to 32 kB. The average latencies for 16 kB and 32 kB cache were reduced by approximately 5% and 30%, compared with the results of 8 kB. The miss rates of *bwaves* with 8 kB and 16 kB cache and *leslie3d* were higher than those of other programs. Therefore, the decrement in the average latencies for those programs was limited according to the Eq. (5).

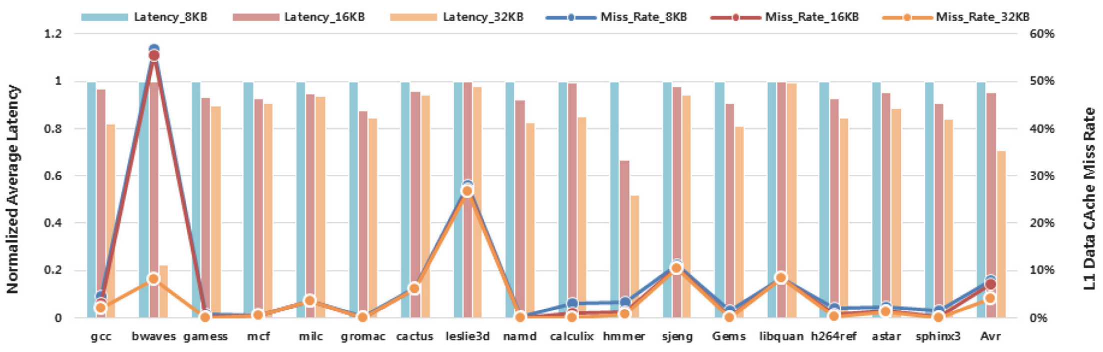


Fig. 6. Normalized average latency and L1 cache miss rate with 8 kB, 16 kB, and 32 kB cache.

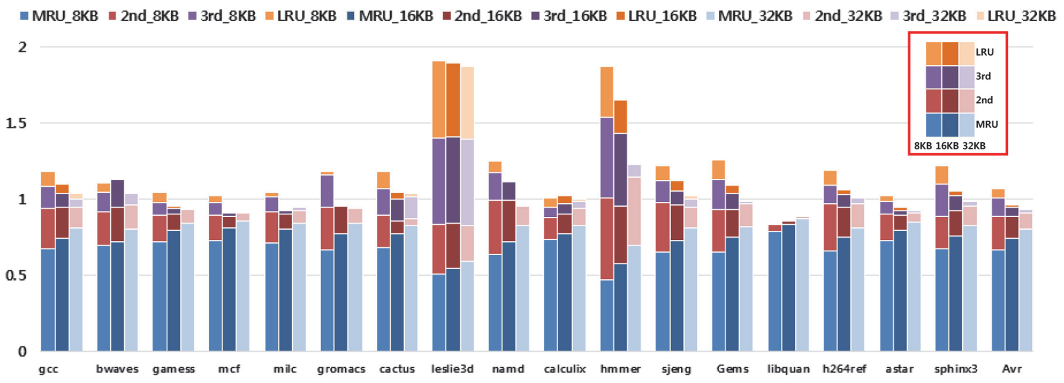


Fig. 7. Normalized dynamic energy consumption. The standard of normalization is the result of the four-cycle fixed latency cache.

C. Normalized Energy Consumption

Fig. 7 presents the dynamic energy consumptions of 8 kB, 16 kB, and 32 kB cache. The standard of normalization is the result of the four-cycle fixed latency cache (4C). There were two main differences between the LRU cache and the CCRU as an aspect of dynamic energy consumption. The LRU cache always enables all tags, while the CCRU searches only the necessary tag information. If the cache block is the third bank, the LRU cache consumes the dynamic energy for all tags and one data, while the CCRU does the dynamic energy for three tags and one data; it skips the fourth tag. Instead, cache blocks including the tag information and the data were moved during cache access. Overall, 6.5% extra dynamic energy was consumed for 8 kB cache, while 3.7% and 6.4% reductions in the dynamic energy were achieved for 16 kB and 32 kB cache, respectively. Therefore, the power dissipation was decreased by 1.3% on average across the cache sizes. These power savings come from that the energy overheads for keeping RU order were compensated by skipping the unnecessary tag information for 16 kB and 32 kB cache, because the access power values for these caches were larger than that of 8 kB.

To investigate these results further, we divided the total energy consumption into the energy consumption for each RU order. The MRU hits consumed 66.4%, 74.0%, and 80.4% of the total energy consumption, while the LRU hits consumed less than 5.9%, 1.6%, and 0.4% with 8 kB, 16 kB, and 32 kB cache, respectively. For *leslie3d* and *hmmer*, the power values were worse than the others; this can be explained by the distribution of cache hit patterns shown in Fig. 3.

VI. CONCLUSION

The cascaded cache based on recently used order can alleviate the drawbacks of the conventional data cache with multiple load-to-use latencies. We separated the whole tag array and data array into small banks. Each cache block was located in the bank according to its recently used order. Whenever a cache block was accessed, all cache blocks in the same set were moved to the corresponding bank. The experimental results show that our proposal achieved a 23.0% average performance gain with 1.3% power savings.

REFERENCES

1. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Cambridge, MA: Morgan

Kaufmann, 2019.
2. R. Alves, S. Kaxiras, and D. Black-Schaffer, “Dynamically disabling way-prediction to reduce instruction replay,” in *Proceedings of 2018 IEEE 36th International Conference on Computer Design (ICCD)*, Orlando, FL, 2018, pp. 140-143.
3. M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, K. Roy, “Reducing set-associative cache energy via way-prediction and selective direct-mapping,” in *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture*, Austin, TX, 2001, pp. 54-65.
4. R. Alves, A. Ros, D. Black-Schaffer, and S. Kaxiras, “Filter caching for free: the untapped potential of the store-buffer,” in *Proceedings of the 46th International Symposium on Computer Architecture*, Phoenix, AZ, 2019, pp. 436-448.
5. R. Alves, N. Nikoleris, S. Kaxiras, and D. Black-Schaffer, “Addressing energy challenges in filter caches,” in *Proceedings of 2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Campinas, Brazil, 2017, pp. 49-56.
6. Arm Developer, “Cortex-R7 processor,” 2021; https://developer.arm.com/ip-products/processors/cortex-r/cortex-r7.
7. C. Kim, D. Burger, and S. W. Keckler, “Nonuniform cache architectures for wire-delay dominated on-chip caches,” *IEEE Micro*, vol. 23, no. 6, pp. 99-107, 2003.
8. M. Rapp, A. Pathania, T. Mitra, and J. Henkel, “Neural network-based performance prediction for task migration on S-NUCA many-cores,” *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1691-1704, 2021.
9. Arm Developer, “Cortex-R7 MPCore Technical Reference Manual,” 2016; https://developer.arm.com/documentation/ddi0458/latest.
10. J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1-17, 2006.
11. J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, “gem5-gpu: a heterogeneous CPU-GPU simulator,” *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34-36, 2015.
12. R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, “CACTI 7: new tools for interconnect exploration in innovative off-chip memories,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1-25, 2017.

Juhee Choi https://orcid.org/0000-0001-6667-8859

Juhee Choi received a B.S. degree in Computer Science from Yonsei University in 2004. He received the M.S. and Ph.D. degrees in Computer Science and Engineering from Seoul National University in 2006 and 2016, respectively. From 2006 to 2020, he worked in the SoC Development Team, at Samsung Electronics Co. Ltd. He is currently an assistant professor at Sangmyung University in the department of Smart Information Communication Engineering. His research interests include SoC design methodology, embedded system design, low-power design, and micro-architecture.

Heemin Park https://orcid.org/0000-0003-4010-232X

Heemin Park received the B.S. and M.S. degrees in computer science from Sogang University, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, Los Angeles in 2006. He was in Yonsei University, Sookmyung Women's University, and Samsung Electronics, South Korea. He is currently an associate professor at Sangmyung University in Cheonan, South Korea, in the Department of Software. His research interests include artificial intelligence, computer vision, cyber physical systems, and pervasive computing.