JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 6, pp. 99-107, June 2020
https://doi.org/10.9708/jksci.2020.25.06.099

Study on the Performance Evaluation and Analysis of Mobile Cache Memory

Sangmin Lee*, Jongwan Kim**, Ji Young Kim***, Dukshin Oh****

*Researcher, AI · Big Data and Software Code Lab, Sahmyook University, Seoul, Korea
**Professor, Smith College of Liberal Arts, Sahmyook University, Seoul, Korea
***Professor, Dept. of Business Administration, Sahmyook University, Seoul, Korea
****Professor, Dept. of Management Information System, Sahmyook University, Seoul, Korea

[Abstract]

In this paper, we analyze the characteristics of mobile cache, which is used to improve the data access speed when executing applications on mobile devices, and verify the importance of mobile cache through a cache data access experiment. The mobile device market has grown at a fast pace over the past decade; however, battery limitations and size, price considerations restrict the usage of fast hardware. Thus, their performance are supplemented by using a memory buffer structure such as the cache memory. The analysis mainly focuses on cache size, hierarchical structure of cache, cache replacement policy, and the effect these features has on mobile performance. For the experimental data, we applied a data set from a microprocessor system study, originally used to test the cache performance. In the experimental results, the average data access speed on a mobile device showed a performance improvement of up to 10 times with the presence of cache memory than without. Accordingly, the cache memory was helpful for the performance improvement of a mobile device when the specifications were identical.

▸ **Key words**: Cache memory, Mobile cache memory, Cache replacement, First In First Out, Least Recently Used

[요 약]

본 논문에서는 모바일 기기에서 앱 실행 시 데이터 접근 속도를 향상하기 위해 사용하는 모바일 캐시의 특징을 분석하고 캐시 데이터 접근 실험을 통해 모바일 캐시의 중요성을 검증한다. 지난 10년간 모바일 기기 시장은 빠른 속도로 성장하였지만, 배터리가 제한적이고, 기기의 크기와 가격이 고려돼야 하므로 속도가 빠른 하드웨어를 사용하기 어렵다. 따라서 캐시 메모리와 같이 메모리 완충 구조를 통해 성능을 보완한다. 본 논문의 주요분석 대상은 캐시 메모리 크기, 캐시의 계층구조 그리고 교체방식과 그에 따른 모바일 성능을 확인한다. 시뮬레이션 데이터는 마이크로프로세서 시스템 연구에서 캐시 성능 확인용으로 사용한 데이터를 사용하였다. 실험결과 모바일 기기에서 캐시 메모리를 사용할 때 데이터에 대한 평균 접근 속도는 캐시 메모리가 없을 때 보다 10배의 성능향상을 보였으며 결과적으로 캐시 메모리는 같은 사양일 때 모바일 기기의 성능향상에 도움이 되는 것으로 나타났다.

▸ **주제어**: 캐시 메모리, 모바일 캐시 메모리, 캐시 교체, 선입선출, 최근 최소 사용

• First Author: Sangmin Lee, Corresponding Author: Dukshin Oh
*Sangmin Lee (sangmin010203@gmail.com), AI · Big Data and Software Code Lab, Sahmyook University
**Jongwan Kim (kimj@syu.ac.kr), Smith College of Liberal Arts, Sahmyook University
***Ji Young Kim (jyk8591@syu.ac.kr), Dept. of Business Administration, Sahmyook University
****Dukshin Oh (ohds@syu.ack.kr), Dept. of Management Information System, Sahmyook University
• Received: 2020. 03. 31, Revised: 2020. 04. 29, Accepted: 2020. 05. 04.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

The market for mobile devices such as smartphones and tablet PCs has been continuously growing [1]. Companies have been making efforts to increase the speed of mobile devices, and lately, the benchmark score [17] of an octa-core CPU [16] of a mobile device has become similar to that of the CPU of regular computers. Since the power consumption, size, and price of mobile device should be taken into consideration [2, 3], high-end devices such as Solid State Drive (SSD) or double data rate synchronous dynamic random-access memory (DDR) cannot be used in mobile devices [3].

When a CPU accesses a data of a main memory or a main memory accesses a data of a secondary storage, the difference of speed between the devices results in a delay. This delay can be overcome by using a cache memory [4].

Cache can be classified into buffer cache and CPU cache. The buffer cache mainly reduces the delay between the main memory and the secondary storage [3, 5], while the CPU cache reduces the delay between CPU and main memory [4].

As the size of cache memory increases, cache gains higher probability for containing the data that CPU requires [5]. Therefore, the performance of mobile devices improves by expanding the cache memory (e.g., using large on-chip cache). However, there is a trade-off relationship between the cache speed and cache size [6] since it gradually takes longer time to search for a data in the cache as the cache memory increases.

When cache memory is full of data, a cache has to replace the data that is no longer needed with replacement policies including the typical replacement policies: LRU (Least Recently Used), Round-Robin (First-In-First-Out, FIFO), Random.

This study analyzes mobile cache and its performance by varying the structures of mobile cache memories including cache sizes and cache levels. Furthermore, we evaluate the performance of LRU replacement policy in mobile devices, which is the policy used in mobile and computer cache, by comparing LRU with Random and FIFO replacement policy.

This study makes the following contributions:

- The importance of cache is analyzed in terms of improving mobile device performance.
- The analysis and simulation results of this study provide practical information to mobile cache researchers.

This paper is organized as follows. Section 2 examines related research on mobile cache memory and replacement policies, and Section 3 describes the experimental methods for the study. Section 4 analyzes the factors affecting the speed of mobile cache through experiments, and finally, Section 5 states the conclusion.

II. Preliminaries

1. Mobile Cache

In mobile devices, low-power RAM and flash memory are used to minimize battery consumption [3, 21]. Since low-power hardware is limited in speed, mobile devices use cache memory to improve the speed of CPU accessing the main memory [1] (Fig. 1).

Information stored in the cache is mainly divided into two types: The first, is the instruction, which is stored when the CPU reads the instruction from the main memory; the second, is the data, which is stored when LOAD and STORE instructions are given. These instructions and data are stored in a unit of block to increase the efficiency of cache memory [7, 22]. If the desired data is in the cache, it is called a hit, and a miss if not; the ratio of hits is called hit rate, and the delay when accessing a memory is called access time [4].

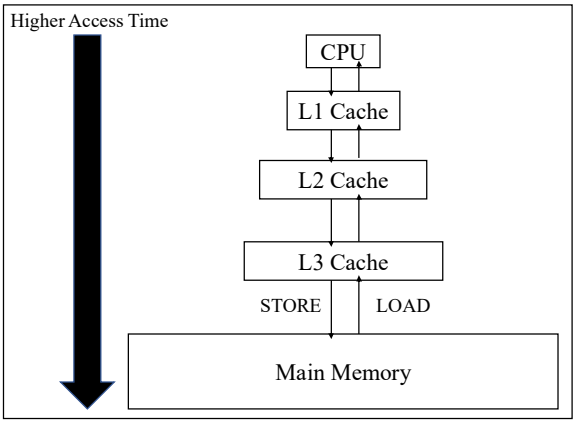

Fig. 1. Structure of a device with a multi-level cache (L1, L2, L3)

The methods for improving the cache performance in mobile devices are as follows. Optimizing the cache memory size, the cache level, the cache line size, the replacement policy, and the data access speed within cache memory are the methods [8, 23]. One way to increase the data access speed within cache memory is to divide the cache into an instruction cache and a data cache [9].

2. Multi-level Cache

A multi-level cache structure is used for the cache memory to maintain its speed because the caching speed slows down when a large cache memory is used [6] (Fig. 1). In a multi-level cache structure, the cache memory size increases level by level to maintain the speed of the upper-level cache [6]. Therefore, the cache of a typical mobile device, smartphone, consists of three level of cache memory namely, L1, L2, and L3 [19], and that of a tablet PC consists of two levels, L1 and L2 [16].

3. Cache Replacement Policies

When there is no space in the cache memory, the existing data must be evicted with a replacement algorithm to store new data. The ultimate goal of all cache replacement algorithms is to maintain the data, which will be used in the near future based on the locality, and to evict the data that has the least necessity. The speed of the device increases depending on the efficiency of the replacements.

In theory, an efficient replacement algorithm is the optimal (OPT) policy, which only evicts the data that will never be used or close to never. Currently, new cache replacement policies are studied, aiming to reach the hit rate of the OPT replacement policy [10].

This study applies the LRU, FIFO, and Random replacement algorithms to the simulation to evaluate the performance.

The LRU replacement policy evicts the data that is least recently used. LRU is known for the superior cache performance to FIFO and Random replacement policy [11]; however, LRU has large overhead costs since LRU requires a process of finding data that haven't been accessed for a long time. In [12], two-queue (2Q), which is an LRU replacement algorithm that supplements the above drawback, is proposed, and as such, novel replacement algorithms based on LRU are often used in mobile devices as well as computers.

The FIFO replacement policy evicts the data that have remained in the cache for the longest time when the cache is full. Since the replacement is done by simply evicting data in chronological order, it does not produce as much overheads as the overheads associated with the LRU.

The Random replacement policy is a method for randomly selecting targets that will be replaced when the cache is full. It is a good replacement policy when the data required by CPU has no pattern because Random replacement policy does not generate overheads unlike other policies.

III. Experiment Methodology and Objectives

Table 1 shows the operating system, hardware specification, programming language, tool for building codes, and experimental data used in the experiments of this study.

Table 1. Experimental Environment

Operating System	Windows 10
CPU	Intel Xeon E5-2630 2.20GHz
RAM	128GB
Programming Language	Python 3.7.2
IDE	Visual Studio Code
Experimental data	1,000,001 addresses, each address locating the data

A CPU, multi-level caches, and a main memory are implemented in the experiments with all necessary data stored in the main memory. All data in the main memory are filled with random values and can be accessed by a matching address. The process starts by searching the required data in the register then in the cache memories, and finally in the main memory until it finds the data. The access time was set to 0.1 seconds (s) for the first-level cache (L1), 0.3s for the second-level cache (L2), 0.7s for the third-level cache (L3), and 10s for the main memory.

Fig. 2 shows a recursive algorithm that searches the required data through all storages, and returns the data to all upper-level storages when the data is found.

There are two inputs to the algorithm: *Address*, which represents the memory address of a data, and *StorageIndex*, which represents an index to indicate the storage to search in. In line 1, the register, the cache, and the main memory are gathered in a list in which a storage is selected through *StorageIndex* (line 2). In line 3, the presence of the data in current storage is checked through *Address*. If the data is found in current storage, it is returned to the upper-level storage (line 11) while incrementing a hit-count if the data is found in a cache memory (lines 5-7). If the data is not found in current storage, the data is retrieved from lower-level storage by repeating the algorithm (lines 8-9). The retrieved data is stored in current storage with the address and returned to the upper-level storage device, thereby storing the retrieved data in the register in the end. The size of a data stored in a memory is set to 1 kilobyte (KB).

```
function LoadData(Address, StorageIndex)
Input:
- Address: an address of data
- StorageIndex: an index for storage type
1    StorageList = [Register, Cache, RAM]
2    CurrentStorage = StorageList [StorageIndex]
3    IF Address ⊆ CurrentStorage
4        Data = CurrentStorage[Address]
5        IF CurrentStorage is Cache
6            HitCount = HitCount + 1
7            AccessTime = CaculateTime(StorageIndex)
8    ELSE
9        Data = LoadData(Address, StorageIndex + 1)
10       StoreData(CurrentStorage, Address, Data)
11   return Data
```

Fig. 2. The cache simulation algorithm

The data set used to check the cache performance from a microprocessor system study [18] has been applied to the mobile cache performance experiment by removing unnecessary data. A data of the original data set consists of *n* and *address*. *n* is a number between 0 and 3 indicating type of the MESI protocol [20] which is a protocol to maintain the consistency of cache data and main memory data. *address* is a 32-bit address that incidates the location of a data in all memories.

In the mobile cache simulation, *n* was removed and only *address* was used to evaluate cache performance by monitoring the data access speed and hit rate.

In the data set, a total of 43,502 different addresses were used, and Fig. 3 shows the distribution of individual address of the data set, total of 43,502 addresses. For example, within the data range of 100,000 to 200,000, mostly addresses between 1 and 10,875 and 10% of addresses between 10,876 and 21,750 were used. In another example, the data ranging from 700,000 to 800,000 are constituted of 60%, 25%, 10%, and 5% of the addresses between 32,622 and 43,502, 21,751 and 32,625, 1 and 10,875, 10,876 and 21,750 respectively. The addresses from 1 to 10,875 are used regularly in the data set and the remaining addresses are used temporarily.

In the experiments, the following three test cases are used to analyze the characteristics of the cache replacement algorithm.

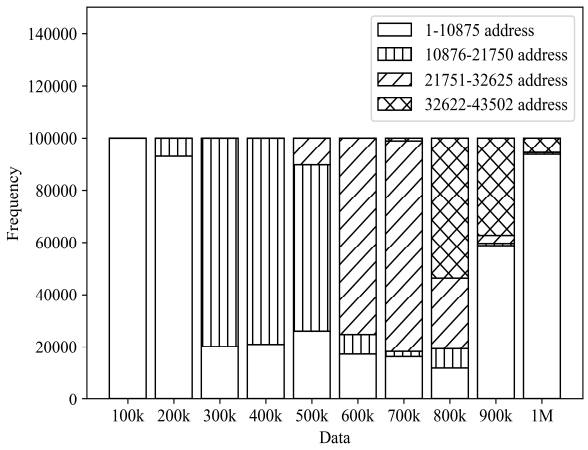

Fig. 3. Data set distribution

[Test Case 1] The size of the cache memory, hit rate, and access speed are analyzed.

The analysis is performed considering that in general, smartphone cache memories consist of 64KB for L1, 1,000KB for L2, and 2,000KB for L3, noted as 64/1000/2000. In other words, we implement different sizes of cache by increasing and decreasing the size to test the effect of cache memory size on hit rate and access time. There are five size variations of L1/L2/L3 implemented: 16/250/500; 32/500/1000; 64/1000/2000; 128/2000/4000, and 256/4000/8000.

[Test Case 2] The efficiency of the multi-level structure of cache memory is evaluated.

The multi-level cache improves the performance of devices compared to single level cache [6, 13]; however, in tablets, the fastest CPU [16] consists of a 2-level cache of 128/8000. Therefore, the effect on the hit rate and the access speed is examined for 2-level, 3-level, and 4-level caches. In the experiment where L3 is removed, 128/4000 and 128/8000 are studied.

In experiment of adding a fourth cache level L4, 128/2000/4000/12000 and 128/1000/2000/12000 are studied by adding a cache size of 12,000KB that takes 3s to access.

[Test Case 3] The performance of LRU replacement policy is compared to other policies.

In the above two experiments, the performance of the standard LRU replacement policy is evaluated by comparing to the results of Random and FIFO replacement policies. It is important to examine the performance of the LRU replacement policy because it is the replacement policy used in mobile devices and computers.

IV. Evaluation of Mobile Cache Performance

1. Cache Size and the Performance

In this section, performance analysis is conducted according to the variation in the cache size, cache hierarchy, and replacement policy. In Table 2, all results of the experiment and the replacement policies exhibiting the best performance according to the structure of the cache are demonstrated. The structures of the cache are expressed as S1, S2, …, S10 in Fig. 4 to Fig. 7 with the precise structures stated below the figures.

The structure frequently used in smartphones is 64/1000/2000. In the experiment of varying cache size, 256/4000/8000 exhibited the best performance among the experimented cache structures.

When the size increased from 16/250/500 to 256/4000/8000, the cache hit rate increased by approximately 1.5 times from 59% to 94% (Fig. 4, Table 2), and the average access time noticeably decreased from 4.2s to 0.7s (Fig. 5, Table 2). With cache memories bigger than 256/4000/8000, hit rate and average access time did not vary significantly. Therefore, it is important to study the appropriate size of mobile cache memory.

When tested without cache, the average access time was 10s, as all data were accessed from the main memory. However, with a cache memory in the mobile device, the access time decreased by up to 1s depending on the size of cache memory. With 64/1000/2000, the average access time was 2s, which was five times faster than the access time in the absence of cache memory.

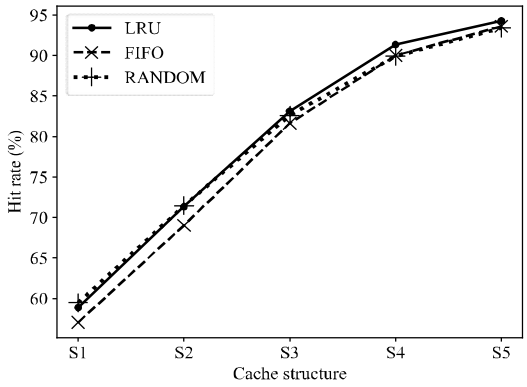

Fig. 4. Hit rate according to cache sizes. The cache structures are displayed below.
S1: 16/250/500, S2: 32/500/1000,
S3: 64/1000/2000, S4: 128/2000/4000,
S5: 256/4000/8000

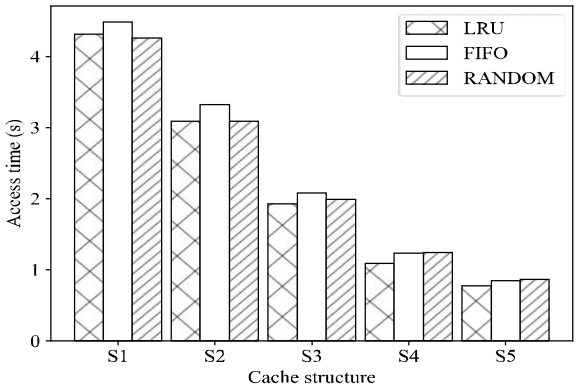

Fig. 5. Access time according to cache sizes. The cache structures are displayed below.
S1: 16/250/500, S2: 32/500/1000,
S3: 64/1000/2000, S4: 128/2000/4000,
S5: 256/4000/8000

2. Cache Hierarchy and the Performance

In tablet PCs, the fastest CPU consists of a 2-level cache with 128KB for L1 and 8,000KB for L2. Therefore, an analysis was conducted to examine the variations in the hit rate and the access time according to addition or removal of a cache in a multi-level cache structure (Table 2). In the experimental results, the hit rate (Fig. 6) and the average access time (Fig. 7) of 128/8000 exhibited comparatively superior performance than 128/4000/8000 which shows why the fastest CPU among mobile devices has a 2-level cache.

In the experiment, a 3-level cache of 128/2000/4000 and a 2-level cache of 128/4000, which both have the same last level cache size, were compared to observe the effect of removing a cache. Based on an LRU replacement policy that showed the best performance, the hit rate did not vary (Fig. 6); however, the access time was about 1.3 times lower for the 2-level cache compared to the 3-level cache (Fig. 7). Thus, the reduction of average access time can be observed if the level of cache is reduced. Note that this decrease in access time is only observed when the size of L2 is equal to or larger than the size of L3 of a 3-level cache while maintaining the access speed. However, since it is difficult to maintain the high speed when the size of cache memory is increased, high technology is required to improve performance when L3 is removed.

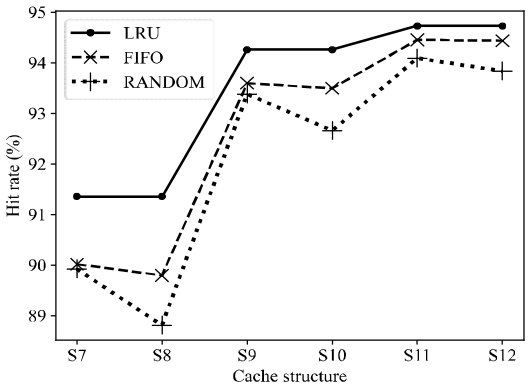

Fig. 6. Hit rate according to cache levels, with some structures including a fourth cache level. The cache structures are displayed below.
S7: 128/2000/4000, S8: 128/4000,
S9: 128/4000/8000, S10: 128/8000,
S11: 128/2000/4000/12000,
S12: 128/1000/2000/12000

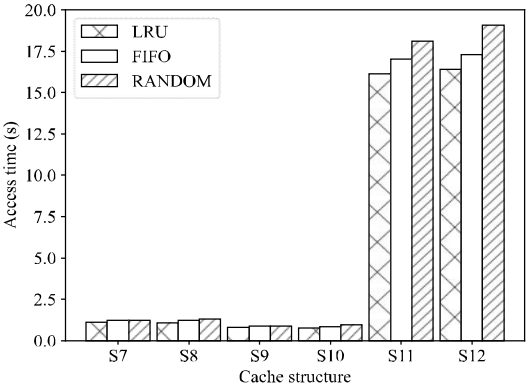

Fig. 7. Access time according to cache levels, with some structures including a fourth cache level. The cache structures are displayed below.
S7: 128/2000/4000, S8: 128/4000,
S9: 128/4000/8000, S10: 128/8000,
S11: 128/2000/4000/12000,
S12: 128/1000/2000/12000

Furthermore, L4, the level 4 cache with a size of 12,000KB which require 3s to access, was examined in the experiment. When 128/2000/4000/12000 was compared to 128/2000/4000, the hit rate increased about 3%, which seemed fair but the average access time drastically increased almost tenfold. However, reducing the size of upper-level caches by half (128/1000/2000/12000) exhibited similar hit rate and access time to 128/2000/4000/12000. Therefore, with the addition of L4, the reduction of upper-level cache size while maintaining the performance is possible. However, with the fact that addition of L4 increases the average access time, the addition of a cache seems meaningless unless the access speed of L4 is increased. Furthermore, cache memory is known to have high power consumption [14, 15] while lowering battery consumption is a important factor on mobile devices.

Table 2. Table describing hit rate and access time according to different cache structures

Cache structure (KB)	Replacement policy	Hit rate (%)	Access time (s)
16/250/500	Random	59.4748	4.260334
	LRU	58.8747	4.307847
	FIFO	57.0499	4.486027
32/500/1000	Random	71.4501	3.089247
	LRU	71.3455	3.090652
	FIFO	69.0393	3.316597
64/1000/2000	Random	82.6045	1.986508
	LRU	83.1376	1.924715
	FIFO	81.6327	2.078204
128/2000/4000	Random	89.9194	1.240775
	LRU	91.3542	1.091746
	FIFO	90.0138	1.227332
256/4000/8000	Random	93.3883	0.866455
	LRU	94.2565	0.771483
	FIFO	93.5962	0.841765
128/4000	Random	88.8066	1.312185
	LRU	91.3542	1.05888
	FIFO	89.7929	1.214273
128/4000/8000	Random	93.3803	0.886838
	LRU	94.2565	0.788966
	FIFO	93.5945	0.860725
128/8000	Random	92.6575	0.938648
	LRU	94.2565	0.777357
	FIFO	93.4919	0.85547
128/2000/4000/12000	Random	94.0897	18.08967
	LRU	94.7254	16.1521
	FIFO	94.452	17.00586
128/1000/2000/12000	Random	93.8332	19.06359
	LRU	94.7254	16.38825
	FIFO	94.4329	17.30579

3. Cache Replacement Policies and the Performance

During the experiments above, the result according to the different replacement policies were compared (Table 2) to verify the fine performance of the commonly -used LRU. In the results, the LRU policy had a higher hit rate and shorter access time compared to the other replacement policies, except for the case where the cache size was too small, making the use of a fine replacement policy meaningless. The LRU replacement policy displayed a 1.17% and 1.02% higher hit rate than the Random and FIFO replacement policy respectively. As the Random replacement policy has a Random replacement characteristic, it had a higher probability of evicting useful, reused data than the LRU replacement policy. The FIFO replacement policy performed worse than the Random replacement policy when the size of cache memory was small, but demonstrated the possibility of an improved perfor mance over the Random replacement policy when the cache size is increased.

V. Conclusions

In this study, experiments were conducted to analyze factors that could increase the performance of mobile cache memory, including cache size, cache level, and the type of replacement policy. Subsequent results proved the importance of cache memories in mobile devices. In the simulation, the average access time was up to ten times lower when the cache size was 256/4000/8000 compared to when there was no cache. Adding an L4 cache level was inefficient owing to the slow access speed and high battery consump tion rate of the cache memory. Removing L3 resulted in superior performance but required high techno logy. LRU was the most efficient replacement policy compared to the FIFO and Random replacement policies, except for the case where the cache size was too small and replacement algorithms were meanin gless.

ACKNOWLEDGEMENT

This paper was supported by the Sahmyook University Research Fund in 2019.

REFERENCES

[1] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and character ization of interactive smartphone applications", IEEE International Symposium on Workload Characterization, In Proceedings of the 2011 IEEE International Symposium on Workload Characterizat ion, No. 11, pp. 81-90, December 2011. 10.1109/IISWC.2011.6114 205

[2] T. Oh, H. Chung, J. Park, K. Lee, S. Oh, S. Doo, H. Kim, C. Lee, H. Kim, J. Lee, J. Lee, K. Ha, Y. Choi, Y. Cho, Y. Bae, T. Jang, C. Park, K. Park, S. Jang, and J. Choi, "A 3.2 Gbps/pin 8 Gbit 1.0 V LPDDR4 SDRAM With Integrated ECC Engine for Sub-1 V DRAM Core Operation", IEEE Journal of Solid-State Circuits, Vol. 50, No. 1, pp. 178-190, January 2015. 10.1109/JSS C.2014.2353799

[3] H. Kim, M. Ryu, and U. Ramachandran, "What is a good buffer cache replacement scheme for mobile flash storage?", Association for Computing Machinery, In Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems, No. 12, pp. 235-246, June 2012. 10.1145/ 2254756.2254786

[4] A. J. Smith, "Cache Memories", ACM Computing Surveys, Vol. 14, No. 3, pp. 473-530, September 1982. 10.1145/356887.356892

[5] S. Jiang, and X. Zhang, "LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache perfor mance", Association for Computing Machinery, In Proceedings of the 2002 ACM SIGMETRICS international conference on Meas urement and modeling of computer systems, No. 2, pp. 31-42, June 2002. 10.1145/511334.511340

[6] J. L. Baer, and W. H. Wang, "On the inclusion properties for multi-level cache hierarchies", Association for Computing Machi nery, In 25 years of the international symposia on Computer architecture, No. 98, pp. 345-352, August 1998. 10.1145/285930 .285994

[7] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache perfor mance and optimizations of blocked algorithms", Association for Computing Machinery, In Proceedings of the fourth international conference on Architectural support for programming languages and operating systems, No. 5, pp. 63-74, April 1991. 10.1145/106 972.106981

[8] A. Asaduzzaman, I. Mahgoub, P. Sanigepalli, H. Kalva, R. Shankar, and B. Furht, "Cache optimization for mobile devices running multimedia applications", IEEE Sixth International Symposium on Multimedia Software Engineering, In Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering, No. 4, pp. 499-506, December 2004. 10.1109/MMSE.2004.34

[9] J. E. Smith, and J. R. Goodman, "Instruction Cache Replacement Policies and Organizations", IEEE Transactions on Computers, Vol. 34, No. 3, pp. 234-241, March 1985. 10.1109/TC.1985.1 676566

[10] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory", Association for Computing Machinery, In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedde d systems, No. 8, pp. 234-241, October 2006. 10.1145/117676 0.1176789

[11] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite", Association for Computing Machinery, In Proceedings of the 42nd annual Southeast regional conference, No. 42, pp. 267-272, April 2004. 10.1145/986537.986601

[12] T. Johnson, and D. Shasha., "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", Morgan Kaufmann Publishers Inc, In Proceedings of the 20th International Conference on Very Large Data Bases, No. 94, pp. 439-450, September 1994. 10.5555/645920.672996

[13] S. Przybylski, M. Horowitz, and J. Hennessy, "Characteristics of performance-optimal multi-level cache hierarchies", Associatio n for Computing Machinery, In Proceedings of the 16th Annual International Symposium on Computer Architecture, No. 89, pp. 114-121, April 1989. 10.1145/74925.74939

[14] P. Panda, G. Patil, and B. Raveendran, "A survey on replacement strategies in cache memory for embedded systems", 2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics, pp. 12-17, August 2016. 10.1109/DISCOVER.2016.7806218

[15] H. Moon, and S. Jee, "An Energy Efficient and High Performance Data Cache Structure Utilizing Tag History of Cache Addresses", The Kips Transactions: Part A, Vol. 14, No. 1, pp. 55-62, February 2007. 10.3745/KI PSTA.2007.14-A.1.055

[16] Wikipedia, Apple A12X Processor, https://en.wikipedia.org.

[17] Passmark, https://www.passmark.com.

[18] Wicker, L2-Cache-Simulator, https://github.com/wicker/L2-Cach e-Simulator.

[19] WikiChip, https://en.wikichip.org/wiki/samsung/exynos/9825.

[20] Wikipedia, https://en.wikipedia.org/wiki/MESI_protocol.

[21] Y. Kim, and Y. Song, "Impact of processor cache memory on storage performance", Institute of Electrical and Electronics

Engineers Inc., In Proceedings of the International SoC Design Conference 2017, pp. 304-305, November 2017. 10.1109/ISOCC.2017.8368908

[22] J. Ahmed, M. Y. Siyal, S. Najam, and Z. Najam, "Multiprocessors and Cache Memory", SpringerBriefs in Applied Sciences and Technology, Fuzzy Logic Based Power-Efficient Real-Time Multi-Core System, November 2017. 10.1007/978-981-10-3120-5_1

[23] H. Mehboob, and H. A. Niaz, "ENHANCE THE PERFORMANCE OF ASSOCIATIVE MEMORY BY USING NEW METHODS", VFAST Transactions on Software Engineering, Vol. 12, No. 3, pp. 49-56, December 2017.

Authors

Sangmin Lee received the B.S in Computer Engineering from National Institute for Lifelong Education, Korea, in 2020. He is a currently a member of AI and Big Data Software Code Lab at Sahmyook University.

Sangmin Lee is interested in machine learning, deep learning and computer architecture.

Jongwan Kim received the Ph. D. degree in Computer Science and Engineering from Korea University, South Korea, in 2007, B. Sc. degree in Business Administration, and M. Sc. in Computer Science and Engineering from Sahmyook University, Soongsil University, respectively. Dr. Kim joined the faculty of the Smith College of Liberal Arts, at Sahmyook University, Seoul, Korea, in 2016. His research interests include Mobile&Streaming Data Management, Location-based Services, Big data and AI.

Ji young Kim received the Ph. D. degree in Economics from Kyungpook National University, Korea, in 2012. He had been a deputy director at Samsung Securities co., In 2012, he moved to The Catholic University of Korea where he worked as professor of economics department. He is currently a professor of business administration at Sahmyook University. His current research interests include pricing mechanism of whether derivatives, financial issues and economic effect analysis by econometrics, He received HyangChon Best Paper Award in 2012 from Kyungpook National University.

Dukshin Oh received the Ph. D. degree in Management Information Systems from Sangmyung University in Korea. He has published many papers in several journals such as International Journal of Computer Science and Network Security, The KIPS Transactions. His research interests include Management/Computer Information Systems, System Analysis and Design, e-Business and e-Learning systems.