International Journal of Control and Automation
Vol.8, No.5 (2015), pp.315-324
http://dx.doi.org/10.14257/ijca.2015.8.5.29

Modification of Cache Inclusion Property for Multicore Systems

Hoyoung Hwang[1] and Hyo-Joong Suh[2]

[1]*Dept. of Multimedia Engineering, Hansung University*
[2]*Dept. of Computer Science and Engineering, Catholic University, Korea*
hyhwang@hansung.ac.kr, hjsuh@catholic.ac.kr

Abstract

The cache coherency protocol guarantees the validity of the cache block by keeping it with the latest updated contents. In multi-level cache memory hierarchy, the inclusion property enforces the lower layer cache to maintain the superset of its higher layer cache contents. The inclusion property has its benefits for cache coherence, but it may waste valuable cache blocks and bandwidth by invalidating the duplicated contents in the higher level cache. In this paper, a new cache contention management scheme is presented with modified inclusion property to avoid unnecessary cache block invalidation.

Keywords: *Cache coherency, Inclusion property, Multicore, Multi-level cache*

1. Introduction

In computer systems, cache memory performs a critical role both for the performance and the energy efficiency. Each program competes for the cache, while the cache misses invoke power and bandwidth consumption by accessing the main memory as well as slowdown of the program. Recently, multiprocessor systems with multi-core processors are widely used for mobile smartphone systems as well as the high performance computer systems. In general, multi-core processors have higher level private cache memory for each core, and lower level shared cache memory commonly used by several cores. For example, Samsung Exynos processors and Qualcomm Snapdragon processor use this type of cache hierarchy. The Exynos 5 dual-core processor uses 32KB Instruction and Data cache and 1MB shared L2 cache, Snapdragon S4 quad-core processor uses 16KB Instruction and Data cache and 2 MB L2 shared cache. Figure 1 shows typical memory hierarchy of a quad-core processor.

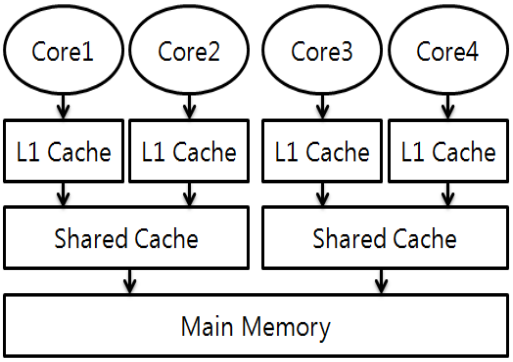


Figure 1. Cache Memory Hierarchy for Multi-processor Systems

The multi-level cache hierarchy causes more complex problems to keep the cache coherence [1]. The snooping protocol, a widely used cache coherence protocol, will need

ISSN: 2005-4297 IJCA

to check whether the same cache block exists in all the cache levels for most cache requests, and this will result in larger overhead and longer delay. To solve this problem, the multi-level cache inclusion property has been used [2]. The cache coherence protocol guarantees the validity of a cache block with latest updated content, and the inclusion property enforces the lower level cache to maintain a superset of its higher level caches [3]. With this inclusion property, we can remove the need to check the existence of a duplicated cache block in higher level cache if the same block is not available on the lower level shared cache. If a cache block is replaced and removed from the lower level shared cache, then the duplicated copy of the cache block on the higher level private cache is invalidated to keep the inclusion property. The multi-level cache inclusion property has its benefits for cache coherence. However, it has the down side effect; it may waste valuable cache blocks duplicated between L1 and L2 caches.

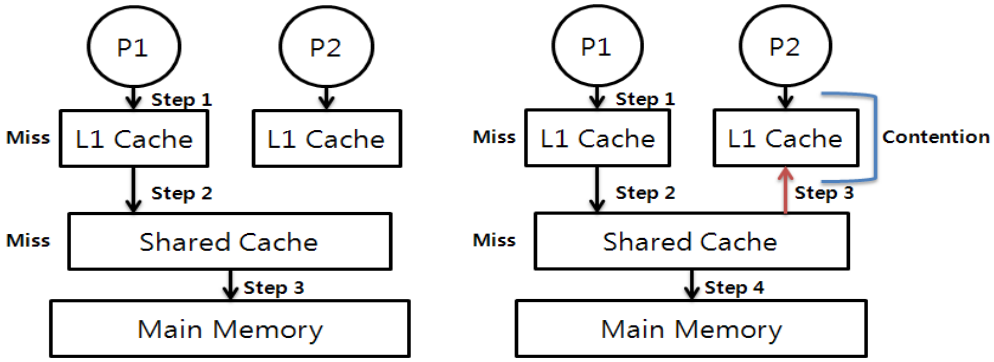
In multicore systems including smartphones, it is common situation to perform the primary processes and other various user applications simultaneously. Each core may perform a different process with hierarchical L1 and L2 cache structure as shown in Fig. 1. Normally, the primary communication process has less frequent requests for cache memory than other application processes that will have frequent requests for cache memory. In that case, a working group of higher priority primary process in L2 cache can be polluted by a lower priority processes with frequent cache requests. The processor with frequent cache requests will increase the possession of shared L2 cache and this will affect the cache performance of the primary process since it will increase the invalidation of L1 cache blocks of the primary process. This kind of cache pollution can be worse to keep the cache coherence and inclusion property.

In this paper, we proposed a new cache contention management scheme to remove the cache invalidation of the primary process by the inclusion property. This scheme classifies the memory blocks into two groups; a shared data group and a private data group, and allows the inclusion property be relieved for the private data group since the cache coherence can be guaranteed for this group even without the inclusion property. This paper is organized as follows. In Section 2, the multi-level cache inclusion property and its problems are discussed. In Section 3, the new cache contention management scheme with modified inclusion property is presented. Section 4 will show the simulation result of the cache performance with benchmark programs for the modified inclusion property, and Section 5 will conclude this paper.

2. Cache Invalidation by Inclusion Property

In multicore systems with hierarchical cache architecture, private L1 cache and shared L2 cache, the purpose of the multi-level cache inclusion property (MLI) is to avoid the high contention for the shared L2 cache and the bottle neck problem caused by it. Figure 2 shows the comparison of memory access with and without multi-level cache inclusion property.

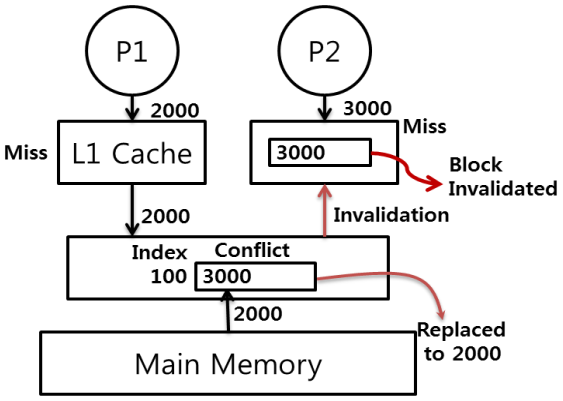
In the MLI case that is shown in Figure 2 (a), the processor 2 (P2) checks the existence of the requested cache block only on the shared cache. If the block is not available on the shared cache, it is guaranteed that the same block does not exist on the L1 cache of the processor 1 (P1). In the non-MLI case as shown in Figure 2 (b), however, the cache miss for the shared cache for a request by P2 does not guarantee that the same block does not exist in the L1 cache of P1. Therefore, the L1 cache of P1 should be checked for all the memory access of P2, which will cause high contention and traffic delay for caches between levels.



(a) Access path of MLI cache (b) Access path of non-MLI cache

Figure 2. Path of Memory Access

In some cases, the multi-level cache inclusion property may cause performance degradation of cache memory. Figure 3 shows the cache contention and invalidation by the multi-level cache inclusion property. The L2 cache requests of P1 for '2000' may replace the cache block '3000' which was used by P2, and by the multi-level cache inclusion property, the duplicated cache block in L1 cache of P2 '3000' will be invalidated. This will cause cache miss on L1 cache of P2. Furthermore, if unbalanced workloads are dispatched on several cores that are sharing the L2 cache, the lightweight core is disturbed by frequent invalidation requests caused by the other heavyweight core to maintain the inclusion property between caches. In this asymmetric workload environment, the multi-level cache inclusion property has a bad effect on the cache performance.



Figre 3. L1 Cache Invalidation by the L2 Cache Contention and Inclusion Property

The inclusion property may cause the waste of bandwidth as well as cache blocks. Each cache block in lower level cache should be duplicated in higher level caches, and a single write operation may require several update operations for the higher level caches. As the depth of the cache hierarchy increases, the waste of cache blocks and bandwidth will be more significant. Therefore, a new policy for the inclusion property should be considered. In prior works, the advantages of not using the multi-level inclusion property have been discussed as follows [6].

- Context switches will be faster because fewer messages will need to be moved up the cache hierarchy for the write backs or invalidation.
- The effective size of the cache system increases by getting rid of data duplication. This is very important when two consecutive levels of caches, namely L1 and L2, or L2 and L3 have similar sizes.

- Conflict misses in the second level cache are reduced due to the fact that heavily referenced blocks are moved to the first level cache, leaving room for other blocks to come in the second level cache.
- Bandwidth can be saved because fewer updates will be needed. The dirty block will not need to be written back to all the higher level caches in the hierarchy until reaching the memory.

Recently, the common use of multicore architecture in smartphones and computer devices increases the need to consider the non-inclusion property. We considered the non-inclusion property for smart devices with multicore architecture and multi-level cache hierarchy, and present a new multi-level cache contention management scheme with modified inclusion property.

3. Cache Contention Management with Modified Inclusion Property

To remove the side effects by the multi-level cache inclusion property, the first step is finding cache blocks which can be writeable among different cores. Executable programs contain several types of binary data. Writeable shared data are keys of the coherency transactions because the other data do not incur major coherency problems. Thus the inclusion property can be limited to the writeable shared data only. For the inclusion control, 5 data types are defined; executable codes, read only data, writable private data, writable shared data, run-time allocated data. The inclusion property can be relieved for the executable codes, read only data, writable private data. Table 1 shows the classification of data block types, and the suggestion of modified control methods for the cache inclusion property.

Table 1. Data Type and Inclusion Property Control

Data types	Inclusion property control
executable codes	Minor control by OS (invalidate when page replacements, etc.)
read only data	
writeable private data	
writeable shared data	Controlled by H/W
run-time allocated data	

According to the data types, every memory block can be grouped as described types when the program loaded into main memory. The type distinction can be processed by the compiler except some run-time allocated data. For simplified processing, the run-time data can be handled the same as the writeable shared data.

To minimize of the inclusion property overhead, it is needed to check each memory access whether it should keep the inclusion property or not. For this, the *inclusion bit* was added which has the information related to the address range of memory. The inclusion bits can be attached to the page table/TLB, and it is controlled by the operating systems. Every virtual address access from a core has to be translated to the physical address with the inclusion attribute. If a translated address does not need to maintain the inclusion property, it can be placed in the shared L2 cache with additional status bit that indicates the data block can be removable without invalidating L1 cache.

Every memory access from a core may invoke a block replacement in the shared L2 cache according to the LRU-like algorithm. If the most aged block is removable, it can be removed without invalidating of the higher level L1 cache. Fig. 4 exhibits

this scenario which does not keep the inclusion property for the L1 data block '3000' thus P2 request results in hit.

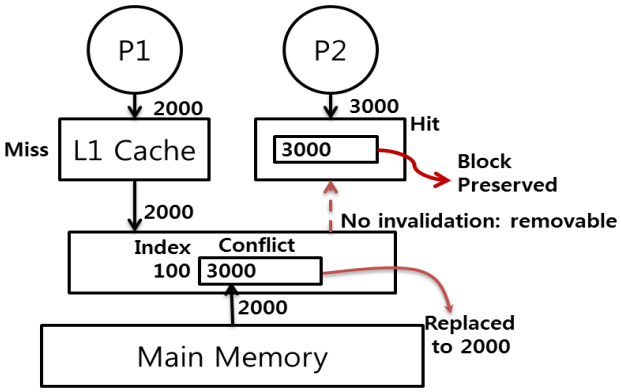


Figure 4. Shared Cache Replacement without Inclusion Property

4. Simulation Result

For the evaluation of the proposed method, The DineroIV [4] cache simulator was used to test the impact of the L1 and shared L2 caches. The DineroIV is a memory trace driven simulator, but does not support the hierarchical shared cache structure of multi-core processor. Thus the simulator was modified as shown in Figure 5 through *trace control program* which handles two memory traces, two L1 caches, and shared L2 cache.

For the memory access traces, CEXP was selected as a lightweight workload that shows high hit ratio for small cache size. There are other 8 programs and a mixed trace. The mixed trace is a memory consuming workload with 507 writes (writeable shared data) from CEXP and other traces. Table 2 shows the property of the workloads from the SPEC'92 benchmark [5].

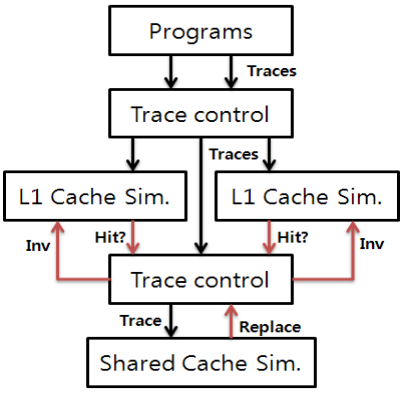


Figure 5. Simulation Flow of Hierarchical Caches

Table 2. Characteristics of the Benchmark Trace Data

Program	Instruction Fetch	Data Read	Data Write	Sum
CEXP	18041	1452	507	20000
COMP	2055	113	356	2524
EAR	4335	711	262	5308
HYDRO	1651	192	284	2127

MDLJD	15931	2065	2004	20000
NASA7	1439	146	270	1855
SWM	15205	1405	3390	20000
UCOMP	2209	161	363	2733
WAVE	2709	254	464	3427
Mixed trace	42240	4722	6962	53924

The L1 cache is assumed to have more than 256B size and tested for 2-way and 4-way. In general, the size of the shared L2 cache should be bigger than the sum of its higher level L1 caches for efficiency, thus the simulated sizes of the shared L2 caches are tuned to 4 times and 8 times of the sum of L1 caches sizes, and tested for 2-way, 4-way, and 8-way.

Table 3 presents the L1 cache miss ratio when CEXP program is performed alone with non-MLI property. According to this result, the CEXP program shows less than 4.06% of cache miss ratio with more than 256B cache size, and even 0% conflict miss for 4-way, and more than 1KB cache sizes.

Table 3. Cache Miss Ratio of CEXP Trace Data, (4 byte/block)

Associativity	**Miss type**	**L1 Cache Size**			
		256B	512B	1KB	2KB
2-way	Miss	4.06%	2.61%	2.54%	2.45%
	Conflict Miss	0.19%	0.15%	0.10%	0.02%
4-way	Miss	3.97%	2.48%	2.45%	2.44%
	Conflict Miss	0.08%	0.02%	0.00%	0.00%

Figure 6 shows the conflict miss ratio of L1 cache with modified cache inclusion property when CEXP program is performed with various cache sizes. This also shows significantly reduced cache miss ratio compared with the traditional way.

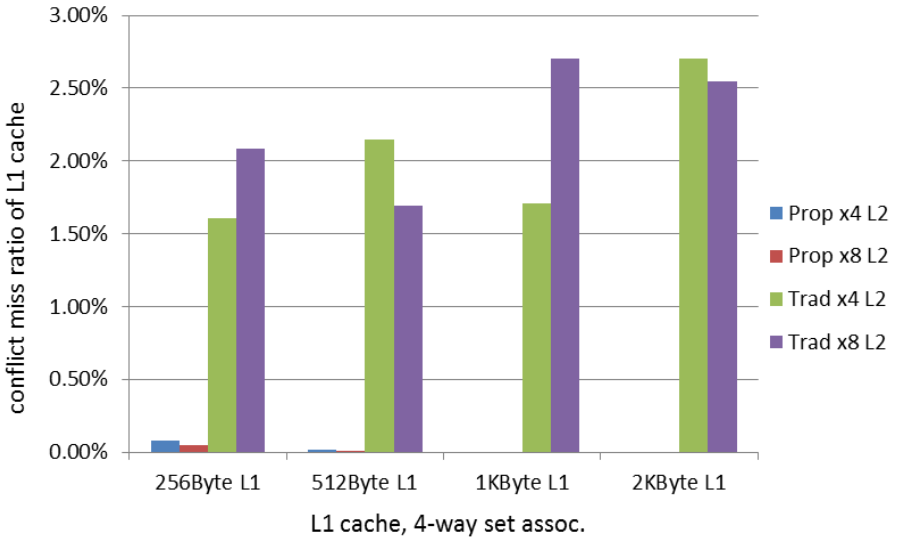
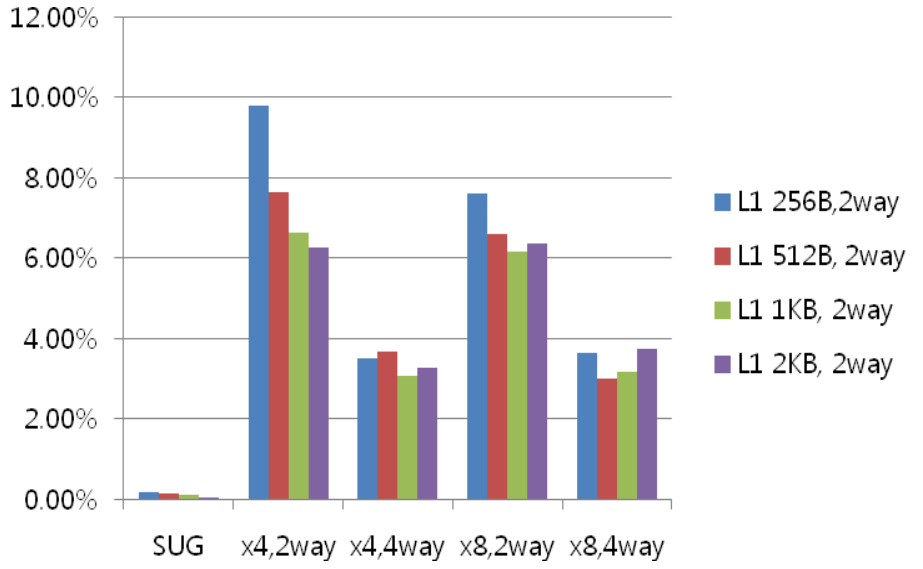


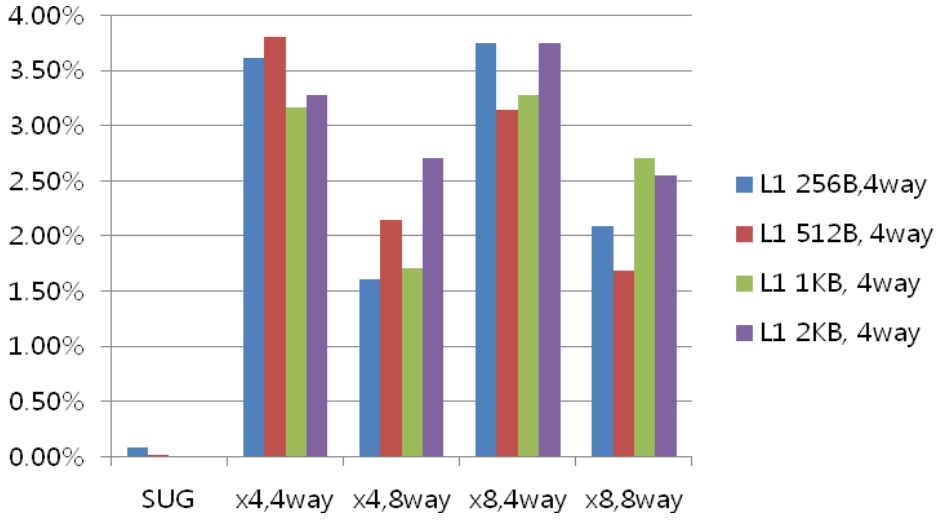
Figure 6. L1 Cache Conflict Miss Ratios with Non-MLI, with the CEXP Trace

Figure 7 shows simulation results of conflict miss ratios originated from the non-inclusive types of data that according to (a) the traditional method with multi-level cache inclusion property, and (b) the proposed method with modified inclusion property.

The simulation results show the inclusion property incurs most of the conflict misses in the L1 cache, and the proposed method with modified inclusion property removes these unnecessary cache misses significantly.



(a) Cache miss ratio with MLI property



(b) Cache miss ratio of Non-MLI property

Figure 7. Conflict Miss Ratio of CEXP Executed Private L1 Cache (x4, x8: shared Cache Size Compared to the L1 cache size)

The simulation results show the inclusion property incurs most of the conflict misses in the L1 cache, and the proposed method with modified inclusion property removes these unnecessary cache misses significantly.

5. Conclusion

Multicore processors with multi-level cache architecture are widely used in mobile smart devices. In those devices, the inclusion property may incur the cache conflict in lower level shared cache, and this will generate unintended block invalidation process up to the higher level cache. With unbalanced workloads dispatched into the cores with shared cache, the cache conflict problem may be worse. The proposed method solved this issue by limiting the inclusion property to the writeable shared data, and by adding the inclusion bit in the page table/TLB and removable bit in the shared L2 cache. The simulation results show that the proposed method reduced the cache miss ratio significantly.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation (NRF) funded by the Ministry of Education (2013R1A1A2057967) of Korea, and the 2015 Research Fund of Catholic University, Korea.

References

[1] M. R. Marty, "Cache Coherence Techniques for Multicore Processors", a PhD Dissertation, University of Wisconsin-Madison **(2008).**
[2] J.-L. Baer and W.-H. Wang, "On the inclusion properties for multi-level cache hierarchies", Proceedings of the 15th Annual International Symposium on Computer Architecture **(1988)**, pp.73-80.
[3] Y. Xie and G. H. Loh, "Scalable shared-cache management by containing thrashing workloads", High Performance Embedded Architectures and Compilers, Springer, **(2010**) pp. 262-276.
[4] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator", http://www.cs.wisc.edu/~markhill/DineroIV/.
[5] K. M. Dixit, "New CPU benchmark suites from SPEC", Proc. 37th IEEE Computer Society International Conference, **(1992),** pp. 305-310.
[6] M. Zahran, K. Albayraktaroglu and M. Franklin, "Non-Inclusion Property in Multi-level Caches Revisited", IJCA, vol. 14, no. 2, **(2007).**
[7] H. Hwang and H.-J. Suh, "A New Contention Management Scheme for Multicore Systems", Advanced Science and Technology Letters, vol. 85, **(2015),** pp. 20-24.
[8] T. Givargis, "Improved Indexing for Cache Miss Reduction in Embedded Systems", Proceedings of Design Automated Conference, **(2003),** pp. 875-880.
[9] K. Patel, L. Benini, E. Macii and M. Poncino, "Reducing Cache Misses by Application-Specific Re-Configurable Indexing", Proceedings of Intl. Conf. Computer Aided Design, **(2004)**, pp. 125-130.
[10] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Processor Architectures", Proceedings of Intl. Symp. Microarchitecture, **(2000)**, pp. 245-257.
[11] P. Petrov and A. Orailoglu, "Towards Effective Embedded Processors in Codesigns: Customizable Partitioned Caches", Proceedings of Intl. Work. Hardware Software Codesign **(2001)** pp. 79-84.
[12] N. Topham, A. Gonzalez, Randomized Cache Placement for Eliminating Conflicts, IEEE Trans. Computers **(1999)** Vol.48, No.2, pp.185-192
[13] A. Gonzalez, M. Valero, N. Topham, J. M. Parcerisa, Eliminating Cache Conflict Misses Through XOR-Based Placement Functions, Proceedings of Intl Conf. Supercomputing **(1997)** pp. 76-83
[14] R. R. Rau, Pseuso-randomly interleaved memory, Proceedings of Intl. Symp. Comp. Arch. **(1991)** pp. 74-83
[15] A. Seznec, A Case for Two-Way Skewed-Associative Caches, Proceedings of Intl. Symp. Comp. Arch **(1993)** pp. 169-178
[16] A. Agarwal, S. D. Pudar, Column-Associative Caches: A technique for Reducing the Miss Rate of Direct Mapped Caches, Proceedings of Intl. Symp. Comp. Arch. **(1993)** pp. 170-180
[17] P. D'Alberto, A. Nicolau, A. Veidenbaum, A Data cache with dynamic mapping. Proceedings of Languages and Compilers for Parallel Computing, LNCS **(2003)** pp.436-440s
[18] M. A. Vega-Rodriguez, J. M. Sanchez-Perez, J. A. Gomez-Pulido, An Educational Tool for Testing Caches on Symmetric Multiprocessors, Microprocessors and Microsystems **(2001)** Vol.25, No.4, pp.187-194
[19] J. Kim, S. Lim, J. Kim, Dynamic Reconfiguration of Cache Indexing in Embedded Processors, IEICE Trans. Information and Systems **(2007)** pp. 637-647

Authors

Hoyoung Hwang, he received the BS, MS, and PhD degree in computer engineering from Seoul National University, Korea. In 2007, he joined the department of Multimedia Engineering, Hansung University now working as an associate professor. His research interests include data communications, mobile ad-hoc networks, multimedia systems, and IoT issues.

Hyo-Joong Suh, he received BS, MS, and PhD degree in computer engineering from Seoul National University, Korea. For 2000-2003, he worked as a senior researcher at GCT Research, Inc. He joined Catholic University in 2003, and now working as a professor. His research interests include computer architecture, memory systems, and embedded systems.