J. Parallel Distrib. Comput. 71 (2011) 1545–1557

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A leakage-aware L2 cache management technique for producer–consumer sharing in low-power chip multiprocessors

Hyunhee Kim, Jihong Kim *

School of Computer Science and Engineering, Seoul National University, Seoul, 151-742, Republic of Korea

ARTICLE INFO

Article history:
Received 26 May 2010
Received in revised form
13 April 2011
Accepted 24 August 2011
Available online 31 August 2011

Keywords:
Chip multiprocessors
L2 cache
Leakage
Producer–consumer sharing

ABSTRACT

This paper proposes a novel leakage management technique for applications with producer–consumer sharing patterns. Although previous research has proposed leakage management techniques by turning off inactive cache blocks, these techniques can be further improved by exploiting the various run-time characteristics of target applications in CMPs. By exploiting particular access sequences observed in producer–consumer sharing patterns and the spatial locality of shared buffers, our technique enables a more aggressive turn-off of L2 cache blocks of these buffers. Experimental results using a CMP simulator show that our proposed technique reduces the energy consumption of on-chip L2 caches, a shared bus, and off-chip memory by up to 31.3% over the existing cache leakage power management techniques with no significant performance loss.

1. Introduction

One of the major concerns in designing modern chip-multiprocessors (CMPs) has been power dissipation that consists of dynamic power and static power. Dynamic power is caused by the transistor switching activity while static power is mainly due to sub-threshold and gate-oxide leakage. Since static power is a major source of total power dissipation as the process technology advances below 65 nm, managing static power consumption has become one of the critical design objectives in realizing low-power CMPs. According to the International Technology Roadmap for Semiconductor (ITRS) [14], one of the key challenges for low-power chips is to manage the leakage power of devices since using low leakage devices can significantly reduce the static power of chips without degrading its performance. Although the emerging technology, such as high-K dielectrics [20], can effectively reduce gate-oxide leakage, subthreshold leakage is still dominant in total leakage power consumption.

In most CMPs, a large on-chip L2 cache is employed to hide the performance gap between processors and main memory. Furthermore, on-chip cache size is expected to increase in future CMPs because more processor cores will demand more data transfers from the main memory. Because of its size, an L2 cache often becomes the major source of the on-chip leakage power among all of the on-chip components. For instance, for a single processor, it is known that 30% of total power dissipation in Alpha 21264 and 60% in StrongARM is consumed by cache and memory structures [19]. As for a multicore processor, in the Niagara-2 processor, an L2 cache consumes more than 20% of total power dissipation while the power consumption of cores accounts for 30%. In [7], simulation results have shown that 37% of total power is consumed by an L2 cache on average when running parallel applications and 97% of this is the leakage power consumption. In addition, the simulation results in [21] have also shown that leakage energy consumption ranges from 80% (2-core 2-issue 8-MB L2) to 30% (8-core 8-issue 1-MB L2) and most of it is consumed by L2 caches. These results emphasize that reducing the leakage power consumption of an L2 cache becomes particularly important for modern CMPs.

One widely used technique for reducing the leakage power consumption in a cache is the cache block turn-off technique. Gated-V_{dd} [23] eliminates the leakage power consumption of a SRAM cell by gating off supply voltage when it is not active. This circuit is employed by the cache decay technique [15], and the cache block is turned off when they are not accessed for a predefined number of time-out threshold cycles until the next miss occurs. In this technique, additional misses occur because the turned-off cache blocks do not preserve data. To overcome the drawbacks of the cache decay, the drowsy cache [11] supplies minimum power to preserve data. Although this technique does not incur extra misses, the access latency of a cache increases, which causes performance degradation. On the other hand, the Adaptive Mode Control (AMC) technique [28] has proposed a method to dynamically change the time-out threshold value by using the run-time characteristics of applications, and use the smaller time-out threshold value if the overall performance does not degrade. This allows cache blocks to be turned off earlier than

* Corresponding author.
E-mail addresses: hh0726@davinci.snu.ac.kr (H. Kim), jihong@davinci.snu.ac.kr, jihong.kim.snu@gmail.com (J. Kim).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.08.006

when the static predefined threshold value is used. However, these techniques are designed for single processors and thus do not consider the characteristics of the cache blocks in CMPs.

The existing leakage management techniques can be further improved by exploiting the characteristics of cache blocks in CMPs. Virtual Exclusion proposed in [12] exploits the multi-level inclusion property (which is an essential property for efficiently implementing cache coherence protocols in multiprocessor systems) in order to save leakage power consumption. It reduces leakage power consumption by turning off repetitive cache blocks in an L2 cache. However, it can be used only when the sizes of L1 and L2 cache blocks are the same. Monchiero et al. [22] propose techniques to save the leakage power consumption of private L2 caches in CMPs by using cache coherence protocols. They propose three techniques (1) that turns off L2 cache blocks when they are invalidated by other processors, (2) that turns off L2 cache blocks only after invalidating the corresponding L1 cache blocks, and (3) that avoids turning off modified cache blocks to diminish performance degradation. In these techniques, although the amount of power consumption is reduced compared to the decay technique, the overall performance is improved instead. The Replication-Aware Leakage Management (RALM) technique [16] also reduces leakage energy consumption of L2 caches in CMPs by selectively turning off cache blocks when other processors replicate them. This technique is useful in reducing energy consumption of read-only replications.

Our proposed technique described in this paper also reduces the leakage power consumption in an L2 cache but focuses on the characteristics of the cache blocks in applications, especially, with the producer–consumer data sharing method which is one of the most commonly used in multi-threaded applications as reported in [9,27]. We propose a leakage-aware L2 cache management technique for producer–consumer sharing, called LAPC, which reduces the leakage power consumption of a private L2 cache in CMPs. Fig. 1 shows an overview of a target CMP architecture with private L2 caches, where a MESI-like snoop-based cache coherence protocol is used. A shared L2 cache could also be an option for the L2 cache organization in CMPs. However, prior research [8,3,24] has shown that a private L2 cache achieves better performance if it is efficiently managed. It chooses a private L2 cache organization based on following reasons. First, a private L2 cache has a shorter access latency than a shared L2 cache since data are located closer to the core while they are spread across an entire shared L2 cache. Second, a private L2 cache can provide performance isolation while threads are interfered in a shared L2 cache. Third, a private L2 cache also has benefits in terms of power consumption. For example, it can be considered as a unit of the power management and can simply be turned off when the core is idle. Finally, CMPs with a private L2 cache organization require a simple on-chip interconnect since only the misses from a private L2 cache access an interconnect, which also consumes less power. Therefore, it does not require the interconnect with high bandwidth, such as the crossbar in Niagara-1 [24], and thus the proposed low-power technique is developed based on this low-power and simple architecture.

In these multiprocessor systems with private L2 cache, when multiple processors share the same memory block, its copies are made in each local private L2 cache. Particularly, in an application where a producer and a consumer threads communicate with each other through shared buffers, because they share the same memory blocks that belong to the shared buffers, the cache blocks for these memory blocks are copied in a producer's and a consumer's caches. When a producer thread writes the produced data into the shared buffer, however, a copy in a consumer's cache, which is part of the shared buffers, is invalidated to maintain cache coherency. It means that, after a consumer thread reads the data from the shared buffer, the shared cache block belonging to the shared buffer in a consumer's cache can be immediately turned off, which is possible if we know which cache blocks belong to the shared buffer.

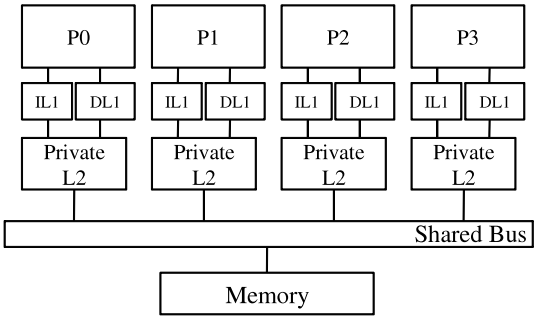


Fig. 1. An overall architecture of a target system.

The proposed technique, first, detects the addresses of the shared buffer that a producer and a consumer thread use to communicate with each other by observing the particular pattern of cache coherence transactions. Second, it adapts an aggressive time-out threshold to the detected shared buffer cache blocks in order to aggressively turn them off. The simulation results show that this simple technique can improve the efficiency of the existing leakage management technique by up to 31.3% with no significant performance loss and negligible hardware overhead. The rest of this paper is organized as follows. In Section 2, we show the motivation of the proposed technique while the LAPC technique is described in detail in Section 3. Section 4 shows the comparison results of the proposed and other existing techniques. Finally, Section 5 concludes with a summary and directions for future work.

2. Motivation

Leakage power consumption becomes a major design issue in realizing low-power microprocessors as the process technology advances. Even though the existing techniques can efficiently reduce the leakage power consumption in L2 caches, they can be improved mainly by exploiting the special characteristics of cache blocks in CMPs. In CMPs, in particular, these characteristics occur when multiple threads communicate with each other by sharing memory space. One of the most well-known data sharing patterns is producer–consumer data sharing where a producer thread writes data into the designated shared buffers and subsequently a consumer thread reads the data written by the producer thread. The proposed technique is based on the characteristics of cache blocks which are observed in producer–consumer applications. In this section, we describe the characteristics of shared buffer cache blocks and explain how they are exploited.

In multiprocessor systems with private L2 cache, several copies of the same memory block are allocated in each private L2 cache when multiple processors share it. Especially in an application with producer–consumer sharing patterns, the memory blocks which belong to shared buffers are shared by a producer's and a consumer's processors, and the copies of the same memory blocks are allocated in each cache. When a producer writes the data into the shared buffer, the corresponding copy in a consumer's cache is invalidated in order to maintain cache coherency. If it is detected that the cache block belongs to shared buffers, the cache block can be immediately turned off after the last read request is issued. The proposed technique aims to aggressively turn off a consumer's cache blocks which belong to the shared buffers without performance loss in order to reduce leakage power consumption.

The proposed technique to detect and aggressively turn off shared buffer cache blocks is based on the following observations. First, when a producer and a consumer access the shared buffer, a particular pattern of cache coherence transactions occurs. Fig. 2

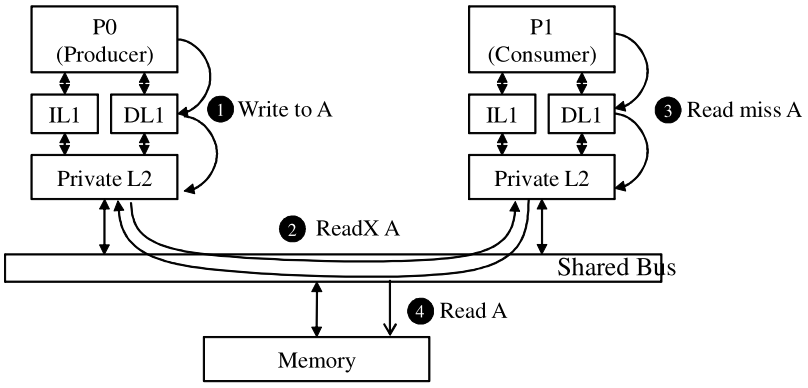
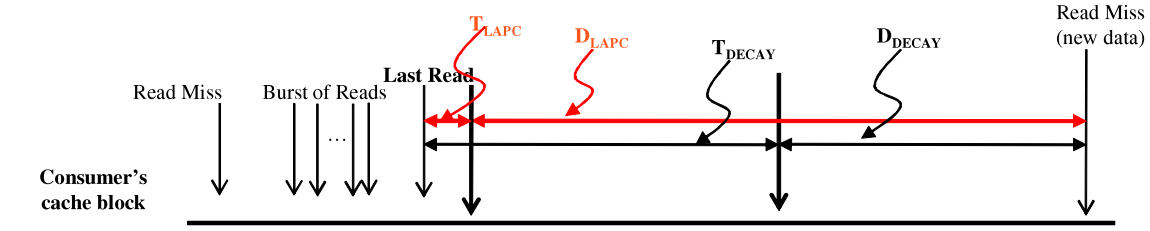
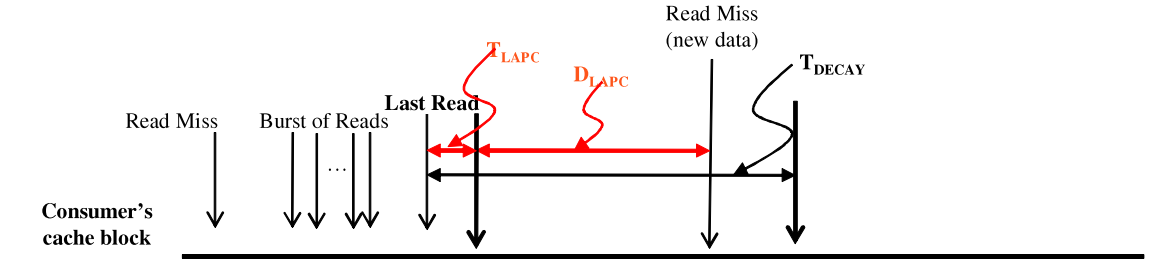


Fig. 2. A series of cache coherence transactions.



(a) The turned-off cache block in DECAY.



(b) The turned-on cache block in DECAY.

Fig. 3. The time-out threshold cycles and dead cycles in DECAY and LAPC.

illustrates a series of cache coherence transactions that occurs when a producer and a consumer communicate through the shared buffer, assuming a snoop-based MESI (Modified, Exclusive, Shared, Invalid) protocol [10]. In this figure, "A" is one of the shared buffer addresses. (1) Every time a producer writes data to the shared buffer, (2) it generates the cache coherence transaction "ReadX A" in order to exclusively read the data by invalidating a copy of the same data in another cache, if it exists. (3) On the other hand, whenever a consumer tries to read the produced data from the shared buffer which is written by a producer, a read miss occurs because the data is invalidated by a producer in (2). (4) a consumer generates "Read A" to read the data from other caches or off-chip memory. These "ReadX" and "Read" transactions occur on the shared bus for all of the shared buffer addresses whenever a producer and a consumer access the shared buffer to communicate. Therefore, if this pattern is detected for a particular address, it might be a shared variable or part of the shared buffer addresses. Since the shared buffers are generally located in consecutive memory space, the addresses of the shared buffers that are present on the shared bus have a high-level spatial locality. In the proposed technique, if consecutive addresses with a pair of "ReadX" and "Read" transactions are detected, they are considered part of the shared buffer.

Second, if the addresses of the shared buffer are known, their cache blocks can be turned off earlier than other cache blocks based on their special access patterns. In a consumer's cache, the cache block belonging to the shared buffer is read from a producer's cache or the off-chip memory every time a consumer tries to read the data written by a producer. In most producer–consumer applications, once a consumer copies the produced data from the shared buffer into its local memory space, it processes its work with the local data. In addition, since the shared buffer has high spatial locality, a burst of the read requests are sent to the cache block when a consumer reads the data from the shared buffer. After the burst, the cache blocks are not accessed any more and invalidated by a producer when the producer modifies the data. It means that we can turn them off immediately after the last request in the burst of the reads is issued in order to reduce the leakage power consumption. It should be noted that this does not incur any performance loss because the data might not be needed again.

Fig. 3 shows the time-out threshold and dead cycles of the cache block belonging to the shared buffer on a consumer's side especially when using the DECAY and LAPC techniques. Fig. 3 (a) shows the case of a cache block that can be turned off while Fig. 3 (b) shows the case of a cache block which cannot be turned off when using the cache decay technique [15]. T_{DECAY} and D_{DECAY} indicate the time-out threshold and dead cycles of the cache decay technique which does not consider the characteristics of the application, respectively. In this technique, if a cache block is not accessed for the T_{DECAY} cycles, it is turned off for the D_{DECAY} cycles until the next miss occurs.

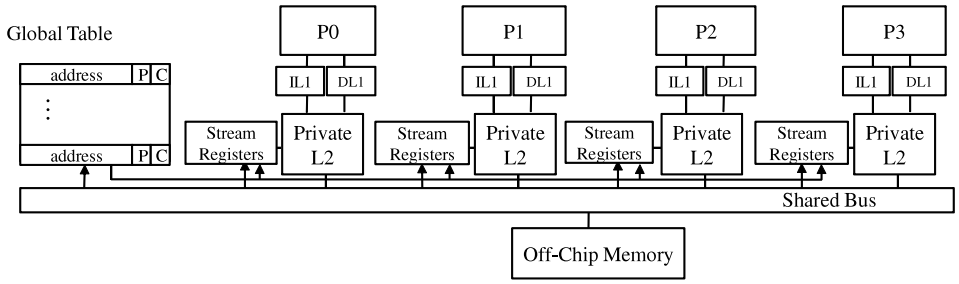


Fig. 4. An overall architecture of LAPC.

We can improve the existing cache decay technique using an adaptive time-out threshold for shared buffer cache blocks. When we use the time-out threshold value without considering any characteristics of cache blocks, they should be long enough to avoid extra misses such as T_{DECAY} in Fig. 3. However, as for the cache blocks of shared buffers, we can turn them off without waiting for the T_{DECAY} cycles as described in Section 2. The proposed technique adapts the time-out threshold value to T_{LAPC}, which is much shorter than T_{DECAY}, only for the cache blocks of shared buffers in order to turn them off aggressively. Therefore, a cache block can be turned off for the D_{LAPC} cycles that are longer than the D_{DECAY} cycles, thereby saving more energy. Furthermore, as can be seen in Fig. 3 (b), a cache block that is not turned off using T_{DECAY} can be turned off if the time-out threshold value is adapted to T_{LAPC}. The amount of the energy reduced by the LAPC technique, $E_{\mathrm{reduction}}$, compared to when using T_{DECAY} as the time-out threshold value for all the cache blocks becomes as follows:

$$E_{\mathrm{reduction}} = \left\{ \sum_{i=1}^{N_{\mathrm{on}}} (D_{\mathrm{LAPC}i} - D_{\mathrm{DECAY}}) + \sum_{i=1}^{N_{\mathrm{off}}} D_{\mathrm{LAPC}i} \right\} \times E_{\mathrm{block\_leak}}, \quad (1)$$

where N_{on} and N_{off} refer to the number of turned-on or turned-off cache blocks among the shared buffer cache blocks when the existing cache decay technique is used while $E_{\mathrm{block\_leak}}$ is the leakage power consumption of a cache block. As for the ith cache block among the N_{on} blocks, it can be turned off for $(D_{\mathrm{LAPC}i} - D_{\mathrm{DECAY}})$ more cycles than the cache decay technique. On the other hand, as for the ith among the N_{off} blocks, it can be turned off for the $D_{\mathrm{LAPC}i}$ cycles even though the cache decay technique cannot turn it off.

Based on these characteristics, the proposed technique detects the cache blocks that belong to the shared buffer memory space and maintains them using only a small hardware. For the detected addresses, it applies an aggressive time-out threshold to reduce the leakage power consumption. In the next section, we explain the technique for detecting shared buffer addresses in detail and show how to apply the adaptive time-out threshold to them.

3. Leakage-aware L2 cache management technique

3.1. Overall architecture

Fig. 4 shows the overall architecture of LAPC which consists of the global table attached to the shared bus and stream registers for each private L2 cache. The global table detects the shared buffer addresses by monitoring "ReadX" and "Read" coherence transactions on the shared bus. If a pair of these transactions is found for a certain memory address, it is considered as one of the shared buffer addresses. The stream registers maintain the detected addresses with a high level of spatial locality. After detecting the addresses of the shared buffer, the proposed technique adapts their time-out threshold value. The detailed operations of the global table and the stream registers are explained in Section 3.2. In Section 3.3, we describe how to adapt the time-out threshold value of the detected cache blocks; the hardware overhead of the proposed technique is calculated in Section 3.4.

3.2. Shared buffer stream detection

The global table maintains entries which have an address and a set of bits representing a producer's (P) and a consumer's (C) processors of the corresponding address. This global table is attached to the shared bus and monitors the cache coherence transactions presented on the shared bus, especially, "Read" and "ReadX". For example, when a producer writes the produced data to one of the shared buffer addresses "0x81280", a "ReadX 0x81280" transaction is generated if the corresponding cache block is shared by other caches. The global table monitors the transaction and is searched to check if there is an entry for "0x81280". If the entry does not exist, a new entry for this address is allocated in the global table. Since this transaction is generated from a producer's cache, the P field of the corresponding entry is set to record a producer's ID. On the other hand, when a consumer tries to read the produced data from the shared buffer, a "Read 0x81280" transaction is generated. When the transaction is detected by the global table, the table is searched to find the entry with the same address. Since a producer has previously allocated an entry for this address, the corresponding entry with the P field set exists. The C field of the entry is set to represent the processor ID of a consumer. When the entry is evicted from the global table, it is inserted into the stream register of a consumer's cache using the C field of the entry.

To efficiently manage the consecutive addresses of the detected shared buffer using a small hardware, we employ the stream registers as in Fig. 5. The stream registers were proposed in [25] for snoop filtering. This technique proposed the stream registers to filter the snoop requests from other caches which are not present in the cache because only a small fraction of snoop requests are actually relevant to the cache. In this technique, the stream registers keep track of the cache blocks in the cache. In particular, it has advantages in applications with a high level of spatial locality because each stream register records address streams using only a small hardware, not each address. In this paper, however, we use them to store the address streams of the shared buffer by exploiting their high spatial locality because the shared buffers are generally located in the consecutive memory space and each reference to the shared buffers occurs sequentially.

Fig. 5 shows each entry of the stream registers and how it operates. Each stream register has several fields which represent a base address, a mask, and a hit interval (HI). The HI fields are added to the original stream registers [25] for our proposed scheme. The base address field has address bits that are common to all of the addresses represented by this stream register. The mask bits are used in deciding whether or not an incoming address matches the base of the stream register. Here, the zero bits of the mask indicate "don't-care" bits. In order to check if the incoming address matches, the base address and the incoming address should be compared after each of them is ANDed with the corresponding mask bits.

When the entry for the address "0x81200, P=0001, C=0001" is evicted from the global table, for example, it is inserted to the corresponding stream registers using the C field of the entry. At first, if there is no entry in the stream register, the new entry is

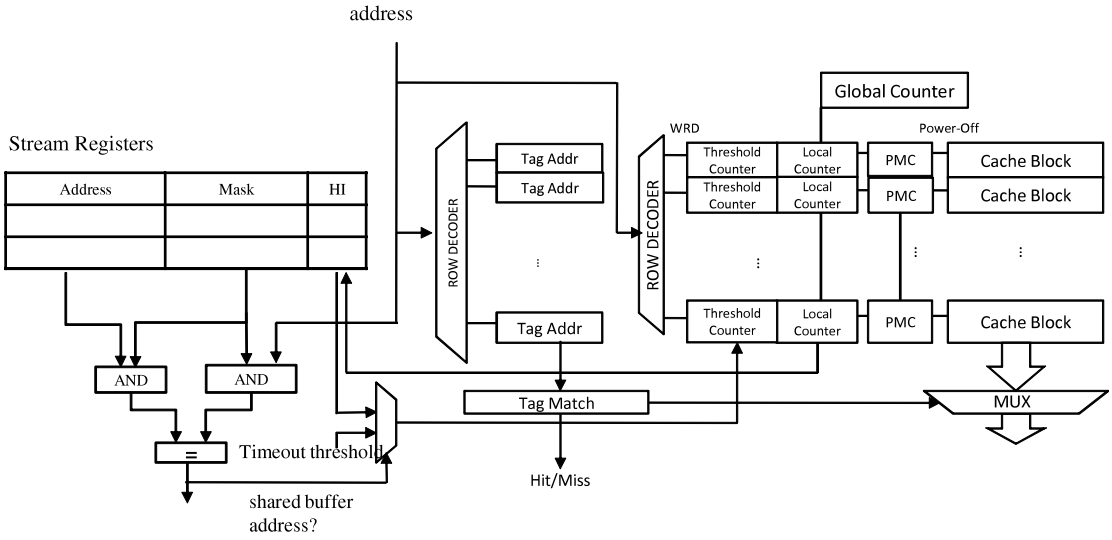


Fig. 5. An overall architecture to adapt the time-out threshold value for the detected shared buffer cache blocks.

allocated in order to store the address, by setting the mask bits to "0xFFFFFFFF". When another entry for the transaction "0x81280, P=0001, C=0010" is allocated in the global table, it is merged with the existing entry only by updating the mask so that a differing bit position between the base and the incoming address becomes zero in the mask, i.e. 0xFFFFFF7F. As in [25], we adopt the "Most Matching Upper Bits" scheme, which selects the entry with the longest matching bits from the high-order bits when deciding which entry to merge with. If there is no empty entry in the stream registers, a FIFO replacement policy is used. If multiple processors access the shared buffer addresses, all of the corresponding P and C bits in the global table entry are set. When the entry is evicted from the global table, the address is sent to all of the consumers using its C bits. This means that the detected addresses are replicated in all of the consumers' stream registers.

The proposed technique applies an adaptive time-out threshold value only for the detected shared buffer cache blocks whose addresses are in the stream registers. For the cache blocks whose addresses are not in the stream registers (because addresses have been evicted from the stream registers or have not been detected yet), a predefined time-out threshold value is applied. An adaptive time-out threshold value for the detected shared buffer is determined based on the hit interval time. The HI field of a stream register entry is updated with the interval time between hits whenever the hit occurs in the corresponding cache blocks. When the hit occurs in the L2 cache, the stream registers are searched to see if there is an entry that matches the hit address. For example, when the address "0x81280" hits in the cache, stream register lookups are performed by comparing the base address and the hit address which are ANDed with the mask. If the results are the same, this means that the hit address belongs to the shared buffer, so its HI field is updated. How to obtain the hit interval time is described in the next section. It should be noted that the incoming address and the stream registers are compared in parallel with the tag lookups, which means that it is not in the critical path.

Our proposed technique does not decrease the overall performance even when a target application is known to have no producer–consumer sharing. When a target application does not have any cache blocks with these patterns, the proposed technique does not detect any addresses and thus does not apply the aggressive turn-off technique. On the other hand, a pair of the ReadX and Read coherence transactions happens for other communication patterns. For instance, any cache block for a shared variable can exhibit this pattern once one processor modifies and then another processor reads it. If this pattern occurs again after the detection, energy consumption can be reduced by turning it off earlier than other cache blocks. However, even when a cache block is turned off by a wrong prediction, but accessed again, data can be brought from another on-chip cache since its copy exists in the cache that generates a ReadX transaction, which means that the penalty of extra miss is small.

When a thread is migrated to another processor, the shared buffer addresses that have been stored in its original stream registers would not be accessed anymore. If the other threads are scheduled and access their shared buffer addresses, new addresses are detected and replace the previous shared buffer addresses. On the other hand, if the thread accesses the shared buffer addresses after migrating, the shared buffer addresses are detected again and stored in its new stream registers. This migration process might slightly decrease the energy reduction by LAPC because the shared buffer cache block could not be aggressively turned off during the migration. However, it does not affect the overall performance.

3.3. Adaptive cache decay implementation

LAPC is based on the time-out based cache decay technique [15]. However, it uses different time-out values for the detected shared buffer cache blocks. A predefined time-out threshold value is used for all cache blocks except for the ones belonging to shared buffers. For the detected shared buffer cache blocks, the time-out value is determined based on the interval time between the hits because they are likely to be invalidated after the burst accesses. By using smaller time-out values than the cache decay technique, LAPC can aggressively turn off the shared buffer cache blocks.

Fig. 5 shows the architecture to adapt the time-out threshold of the cache block belonging to the shared buffer using the stream registers. Similar to the cache decay technique, the proposed technique also keeps track of the elapsed cycles from the last access by using two levels of counters, global and local ones. The global cycle counter sends the tick to the local counters every preset number of cycles and the local counter of each cache block is incremented whenever it gets a tick from the global counter. We also use a threshold cycle counter in addition to the local counter for each cache block in order to implement the adaptive decay technique. The cache block is turned off when the local counter reaches the threshold cycle counter value. As shown in the figure, Power Mode Control (PMC) is used to turn off the cache block by switching the supply voltage to 0 V. When the cache

block is accessed, its supply voltage is switched to 1.0 V to turn it on. Meanwhile, the local counters are set to zero every time the corresponding cache block is accessed in order to keep track of the number of cycles since the last access. In this evaluation, the global counter sends a tick to the local counters every 1000 cycles and the predefined threshold value is 4 million cycles, which is used for the L2 cache in [15].

The proposed scheme is also based on the private L2 cache organization that uses the inclusion property [10], which is usually used in multi-level caches to efficiently implement a cache coherence in multiprocessor systems. To correctly maintain cache coherence and the inclusion property while employing the cache decay technique, the corresponding L1 cache blocks are invalidated when an L2 cache block is turned off, which might cause performance degradation. However, both the cache decay and the proposed techniques turn off L2 cache blocks only when the L2 cache blocks are expected not to be reused in the near future. They determine an L2 cache block is not likely to be reused soon if it is not accessed during predefined time-out threshold cycles. Our heuristic also indirectly assumes that the corresponding L1 cache blocks are also unlikely to be reused in the near future, thus L1-block invalidations do not significantly degrade the system performance. When turned-off cache block is needed again, the missing data is brought from off-chip memory or another core's private L2 cache if the cache has the data while turning on the cache block. Since turning on the cache block can be completed before the missing data is brought into the local private L2 cache, the penalty is the same as additional cycles caused by extra misses.

When a hit to the cache block that belongs to the shared buffer occurs, the HI field of the corresponding stream register entry is set to the value of the local counter. On the other hand, when a miss in the cache block that belongs to the shared buffer occurs, the threshold counter of the corresponding cache block is set to the doubled value of the HI field of the matched stream register while bringing the data from the off-chip memory. If the missed address is not one of the shared buffer addresses, the corresponding threshold counter is set to the predefined threshold value.

3.4. Implementation overhead

To implement the proposed technique, we use a global table with 1024 entries, which is large enough to detect the addresses of the shared buffer. We show that the evaluation results by varying the size of the global table in Section 4. Each entry has an address field with 25 bits and producer and consumer fields with 4 bits, respectively, assuming CMPs with 4 processors. In addition, each private L2 cache has 16 stream registers each of which has a 25-bit base address, a 25-bit mask, and a 5-bit hit interval field. Since the global table records only the block address of the transactions, 25 bits for the base address and 25 bits for the mask are used if the block size is 128 bytes. 5 bits for the hit intervals are enough because the hit to the cache line belonging to the shared buffer occurs with a very short interval.

For the proposed technique, we add 12-bit threshold counters to store adaptive time-out threshold values in addition to the local counters that the existing cache decay technique employs. The threshold cycle counters and the local counters should be longer than the hit interval ones because the former counters record the time-out threshold value and the latter ones keep track of the elapsed cycles from the last access. Therefore, in the proposed technique, the implementation requires a total storage overhead of about 28 KB (4224 bytes for the global table + 440 bytes for all steam registers of the 4 processors + 24576 bytes for threshold counters of four 512 KB private L2 caches). Area overhead of LAPC is also estimated using CACTI 6.5 [5]. The area of the global table and the stream registers is about 0.02 mm^2 while the area of 4 private 512 KB caches is 6.70 mm^2, and thus the area increased by the additional structures is less than 0.3% of the total L2 cache area. As in [15], there is also a 3% area increase for the cache cells because of the logic used to turn off the cache block and the local counter at each cache block. Although the proposed technique assumes 32-bit addresses, it can be easily extended to 64-bit address systems by increasing the addresses and the mask bits stored in a global table and stream registers to 64 bits. This increases the area overheads of these additional structures to twice of the current overheads, which are still negligible.

Table 1
Architectural parameters for the simulation.

Architectural parameter	Specifications
Processors	4 Processors, in-order
L1 I/D caches	private, 16 KB, 2-way, 32 bytes blocks
L2 unified cache	private, 512 KB, 4-way, 128 bytes blocks
L1 cache latency	1 cycles
L2 cache latency	8 cycles
Off-chip memory latency	300 cycles

Table 2
Power/energy parameters for the CMP simulator.

Power/energy parameter		Specifications
CMOS process technology		45 nm
Private L2 cache	Dynamic read energy/access	0.12 nJ
	Leakage power/block	0.098 mW
Global table	Dynamic energy/access	0.01 nJ
	Leakage power	4.10 mW
Stream register	Dynamic energy/access	0.19 pJ
	Leakage power	5.2 uW
Shared bus	Dynamic energy	0.006 nJ
	Leakage power	33 mW
Off-chip memory	Read/write dynamic energy	2 nJ
	Standby power	50 mW

4. Evaluation

4.1. Simulation configuration

We evaluated the proposed technique using a multiprocessor simulator [17] which implements a snoop-based MESI protocol using a shared bus. The proposed technique can be easily applied even to the systems with MOESI because it is orthogonal to a coherence protocol. Shared buffer addresses are detected by their coherence bus transactions such as ReadX and Read, and a pair of these bus transactions also occurs for the shared buffer addresses even though MOESI is used. Table 1 shows the architectural parameters used for the evaluation. We use in-order processors adopted in the multiprocessor systems which prefer simple multiple processors for low-power consumption, such as netbooks, tablet PCs, and smart phones, while simultaneously achieving high-performance through TLP. Many of these high-end embedded systems employ the low-power architectures because they are battery-operated systems. For example, Intel's Atom and ARM11 MPCore, which are the representative high-end embedded multiprocessors, employ an in-order pipeline for their low-power consumption. We obtained power parameters for an L2 cache from CACTI 6.5 [5] using the 45 nm technology and off-chip memory from [6]. The power consumption of a shared bus is also obtained using the bus power model in [1] based on the ITRS. Table 2 shows the power/energy parameters used in the evaluations.

4.2. Benchmark applications

We evaluated the proposed technique using manually composed multi-threaded programs and general multi-threaded applications. For the manually composed 4 multi-threaded programs,

Table 3
Four benchmark combinations used in the experiments.

Benchmark name	Combinations
bench1	LU, FFT, G721, encblowfish
bench2	FFT, MMUL, LU, QSORT
bench3	IFFT, ADPCM, decblowfish, QSORT
bench4	LU, MMUL, IFFT, QSORT

we use bench1, bench2, bench3, and bench4, whose threads communicate with each other through shared buffers. Since we can easily change the sizes of shared buffers in these benchmark programs, we can evaluate how the sizes of the shared buffers affect the proposed technique using these benchmark programs. For these programs, we selected 9 sequential programs from mediabench [18] and mibench [13] and combined four of them into one multi-threaded program. The selected sequential programs are LU, fast Fourier transform (FFT), inverse fast Fourier transform (IFFT), matrix multiplication (MMUL), ADPCM, blowfish data encryption (encblowfish) and decryption (decblowfish), g721 speech compression (G721), and quick sort (QSORT), which are representative embedded applications in audio, video, and digital filtering processing and security areas. Table 3 summarizes the detailed benchmark combinations for bench1, bench2, bench3, and bench4, used in the evaluations. In these combinations, each sequential program is mapped to each core as a thread and executed in a pipelined fashion. Every thread, except for the first and the last threads, keeps two shared buffers. It reads data from one shared buffer and performs its work with the data it has read. After finishing its work, it writes results to another shared buffer that its neighbor thread reads from. Therefore, the input set of each thread is the data read from the shared buffer. With these input data, a thread begins its work after every 32 KB data is read from its shared buffer, and this process is repeated. These pipelined styles are typically used for streaming or signal processing applications. Since each sequential program has different characteristics (e.g., different IPCs, different execution times, different working set sizes, etc.), the evaluations with these combinations can demonstrate that the proposed technique efficiently works in the tasks with various characteristics. For these benchmarks, the default shared buffer sizes are 32 KB. However, with these manually combined benchmarks, a more detailed experiment can be done by varying the sizes of shared buffers in order to see how they affect energy reduction in LAPC, which is shown in Section 4.5. Table 3 shows the four benchmark combinations, used in the experiments.

We also evaluate the LAPC technique with general multi-threaded applications with and without producer–consumer sharing patterns. In order to demonstrate that the proposed technique can efficiently reduce energy consumption in the applications with producer–consumer sharing patterns and does not affect energy consumption and performance significantly even for applications without producer–consumer sharing patterns, we evaluated the applications which show high, medium, and low degrees of producer–consumer sharing patterns as analyzed in [2]. Since these benchmark applications have different degrees of producer–consumer sharing patterns, they can be considered as representatives of the spectrum of real applications. For the applications with the high degree of producer–consumer sharing patterns, dedup from PARSEC [4] and water from SPLASH2 [26] are used. For the application with the medium degree of producer–consumer patterns, volrend, fmm, ocean, and barnes from SPLASH2 are used. Finally, for the applications without producer–consumer patterns, radix from SPLASH2 is included. dedup uses a pipelined programming model in the same way as the manually combined benchmarks. Each thread communicates with another thread through shared buffers. water, volrend, fmm, ocean, and barnes do not employ shared buffers, but have single producer-multiple consumer sharing patterns: one of the threads writes data to the address spaces shared by multiple threads, and other threads read the data from them. On the other hand, radix has no producer–consumer sharing pattern. As for the input set of dedup from PARSEC, *input_simmedium* is used while, as for water, volrend, fmm, ocean, barnes, and radix, *512 molecules*, *head-scaleddown2*, *16-K particles*, *130x130 grids*, *8-K particles*, and *1-M integers* are used as their input sets, respectively.

4.3. Evaluated schemes

In this paper, 5 schemes, DECAY, AMC, SEL_DECAY, DECAY + LAPC, and AMC + LAPC, are evaluated. DECAY and AMC employ the cache decay technique in [15,28], respectively. The DECAY technique uses the predefined value of the time-out threshold and applies it to all cache blocks. The AMC technique dynamically changes the time-out threshold cycles at runtime. In each period, it keeps track of the number of ideal misses that occur when the cache decay technique is not applied as well as the number of sleep misses that are caused by the cache decay technique. When the number of sleep misses becomes much larger than that of ideal misses, AMC increases the time-out threshold value for the next period. In the same way, it decreases the time-out threshold value when the number of sleep misses becomes much smaller than that of ideal misses. In most cases, AMC can reduce energy consumption more than DECAY because it can employ a smaller threshold value when its performance degradation is not significantly large. However, it also applies the same time-out threshold value to all of the cache blocks while the proposed technique can apply different time-out threshold cycles for shared buffer cache blocks.

DECAY + LAPC and AMC + LAPC are the techniques that apply the LAPC technique to the DECAY and the AMC techniques, respectively. In these two techniques, the smaller time-out threshold value is applied to the detected shared buffer blocks, but the time-out threshold values of DECAY and AMC are applied to other cache blocks. The SEL_DECAY technique [22] also turns off cache blocks in the same way as the DECAY technique while modified blocks are selectively turned off in order to avoid performance degradation. In addition, it turns off cache blocks when they are invalidated by a cache coherence transaction as well.

4.4. Energy/performance evaluation

Fig. 6 shows the energy consumption of each technique which is normalized to the energy consumption when no cache leakage management technique is used. We consider both dynamic and leakage energy consumptions of L2 caches, a shared bus, and off-chip memory. In addition to the energy consumption of regular cache and off-chip memory accesses, the overall energy consumption includes the dynamic energy consumption of extra accesses to L2 caches, a shared bus, and off-chip memory. These extra accesses are caused when cache blocks are turned off early due to the proposed technique or the cache decay technique but are reused later. When the turned-off blocks are reused, the missing blocks are brought from off-chip memory or another processor's cache if the cache has the data. As the extra misses increase the execution time, the total leakage energy consumption also increases. The additional leakage energy consumption caused by the increased execution time is also considered. We take into account the dynamic and the leakage energy consumption overheads of the additional structures for the proposed technique using the parameters obtained from CACTI 6.5 as shown in Table 2 as well.

By exploiting the characteristics of the cache blocks in CMPs, DECAY + LAPC can reduce energy consumption by an average of 60.0% and 10.8% over baseline and DECAY, respectively. Although

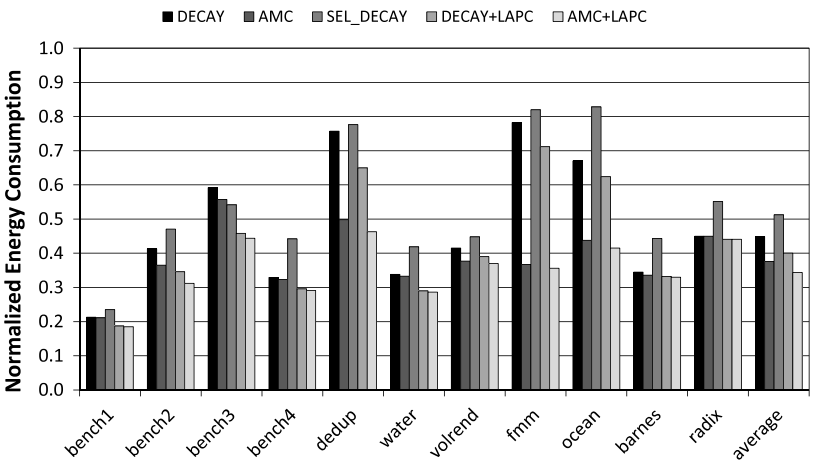


Fig. 6. Normalized energy consumption.

the AMC technique can substantially reduce energy consumption compared to the DECAY technique, it can be more improved by the LAPC technique, and thus AMC + LAPC reduces the energy consumption by 65.7% and 8.6% on average over baseline and AMC, respectively. In particular, for the manually combined benchmarks (`bench1–bench4`), DECAY + LAPC and AMC + LAPC reduce energy consumption by 14.6% and 13.8%, respectively, on average in comparison with DECAY and AMC.

`bench1` shows smaller energy reduction than other benchmarks because G721 in this set has relatively larger execution time than other benchmarks, and thus increasing the turned-off period of the cache blocks by the cache decay technique. Therefore, the ratio of the energy reduction of the LAPC technique to that of the DECAY technique decreases. On the other hand, the greatest reduction in energy consumption compared with DECAY appears in `bench3`. IFFT, ADPCM, and QSORT in this program have the relatively higher utilization of the caches than the other programs, which means that some of the cache blocks are replaced before they are turned off by the cache decay technique. The LAPC technique can aggressively turn off these cache blocks before they are replaced, thus increasing the amount of energy reduction. In this benchmark, DECAY + LAPC and AMC + LAPC improve their energy consumption by up to 22.6% and 20.3% over DECAY and AMC, respectively, because the LAPC technique can further decrease the time-out threshold value for the detected shared buffer cache blocks without no significant performance loss. This indicates that the time-out threshold value cannot be reduced further for shared buffer cache blocks even by the AMC technique, although they can be immediately turned off without performance loss. The amount of energy reduced by LAPC, which is $E_{\text{reduction}}$ in the Eq. (1), compared to the AMC technique becomes smaller. The AMC technique reduces T_{DECAY} compared to when the DECAY technique is used by dynamically changing it depending on benchmarks. This increases D_{DECAY} in the AMC technique, and thus $E_{\text{reduction}}$ of AMC + LAPC over the AMC technique becomes smaller compared to that of DECAY + LAPC over the DECAY technique.

The proposed technique reduces the energy consumption of general parallel benchmarks with producer–consumer sharing patterns as well. For the applications with the high degree of producer–consumer sharing patterns, DECAY + LAPC improves energy consumption by 49.8% and 14.2% compared to baseline and DECAY. In particular, as for `dedup`, the LAPC technique reduces energy consumption by 35.0% and 14.1% over baseline and DECAY, respectively. In this benchmark, each thread is executed in a pipelined fashion where it repeatedly performs its work with the data which it has read from shared buffers and writes results to shared buffers after finishing the work. This is the same way in which `bench1–bench4` benchmarks are executed. As in the `bench1–bench4` benchmarks, DECAY + LAPC reduces energy consumption by detecting shared buffer cache blocks and aggressively turning them off after a consumer thread reads them. The AMC technique also shows significant energy reduction compared to the DECAY technique since it uses different time-out threshold values for each private L2 cache as well as dynamically changing the value while the DECAY technique employs the same value for all private L2 caches during runtime. However, AMC + LAPC can reduce energy consumption by 7.0% and 14.1% more than the AMC technique for `dedup` and `water`, respectively, by immediately turning off the detected shared buffer cache blocks. Even though `water` does not use a pipelined programming model, the LAPC technique detects consecutive addresses that exhibit producer–consumer sharing patterns and aggressively turns them off.

For the applications with the medium-degree producer–consumer sharing patterns, DECAY + LAPC can reduce the energy consumption by 45.8% and 6.1% over baseline and DECAY. These applications have consecutive addresses with producer–consumer sharing patterns, but they are not as many as in the ones with the high degree of the patterns, resulting in smaller energy reduction. However, for `fmm`, DECAY + LAPC can reduce energy consumption by up to 9.0% in comparison with the DECAY technique. AMC + LAPC reduces energy consumption by 3% over AMC because AMC can sufficiently reduce energy consumption over DECAY. This energy reduction is at the cost of performance, i.e., AMC turns off cache blocks too early, leading to the large number of extra off-chip memory accesses. `radix` that has the low degree of producer–consumer sharing patterns shows less energy reduction than the other benchmarks. Nevertheless, the LAPC technique shows about 2.0% of energy reduction by detecting shared variables with these patterns although the amount of reduction is small. In this case, the mask of the corresponding stream register becomes 0xffffffff because it only represents one cache block.

On the other hand, the amount of the energy reduction in the SEL_DECAY technique is about 80% of the DECAY technique as reported in [22] since it selectively turns off modified cache blocks to avoid invalidating the corresponding L1 cache blocks. As a result, it can improve the overall performance compared to other techniques. Although it turns off L2 cache blocks when they are invalidated by another processor, the amount of energy reduction is still less than the LAPC technique. The LAPC technique can turn off shared buffer cache blocks before they are invalidated if the cache blocks are identified as one of the shared buffer cache blocks.

Fig. 7 shows the performance of the 5 techniques which are normalized to the baseline when no leakage management

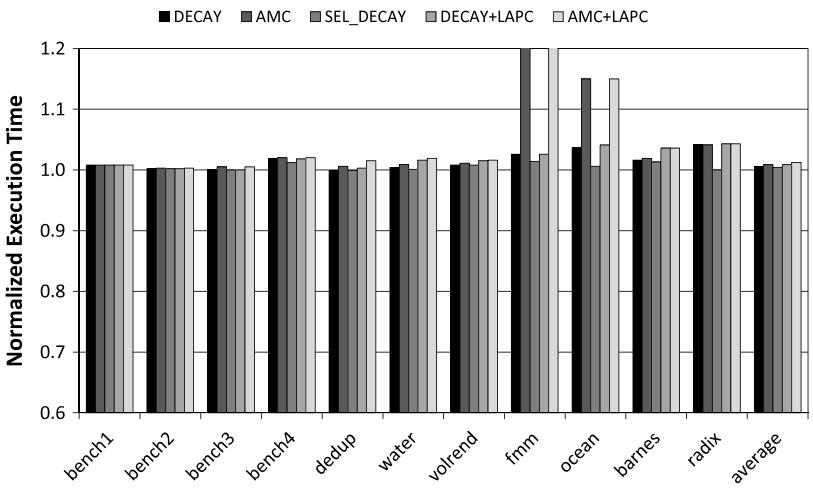


Fig. 7. Normalized performance.

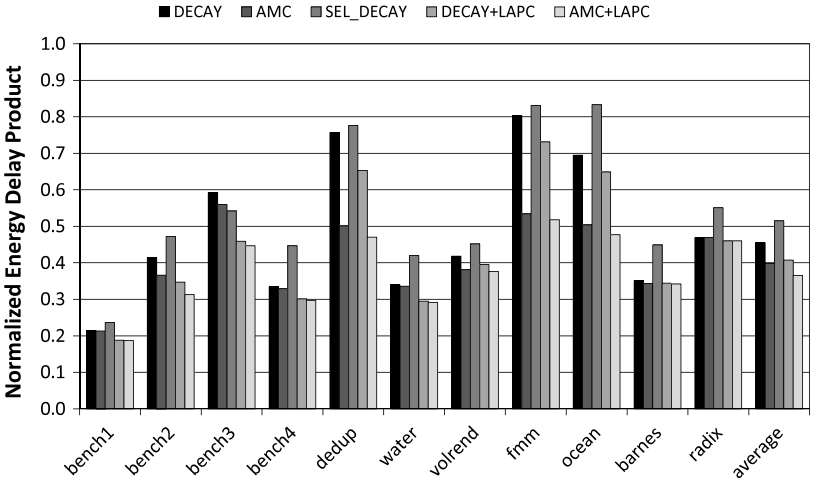


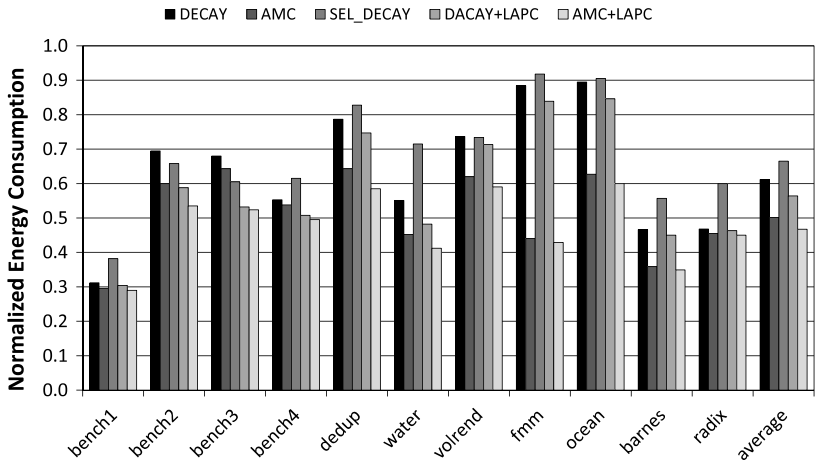
Fig. 8. Normalized energy delay product.

technique is employed. As can be seen, when the LAPC technique is applied to the DECAY and the AMC techniques, i.e., DECAY + LAPC and AMC + LAPC, it incurs no significant performance loss because the cache blocks of shared buffers are not required after being turned off, which means that extra misses caused by turning them off aggressively do not occur. This also indicates that our shared buffer detection technique does not cause a significant performance loss even though the stream registers detect more addresses than the shared buffers used. When the turned off cache blocks are reused due to a wrong prediction, they can be brought from another processor's cache if the cache has its copy, thereby reducing the penalty of extra misses. The normalized energy delay product is shown in Fig. 8. Since the performance degradation caused by the LAPC technique is not significant, energy delay product is similar to energy consumption. DECAY + LAPC and AMC + LAPC reduce energy delay product by 10.4% and 8.2% compared to the DECAY and the AMC techniques, respectively.

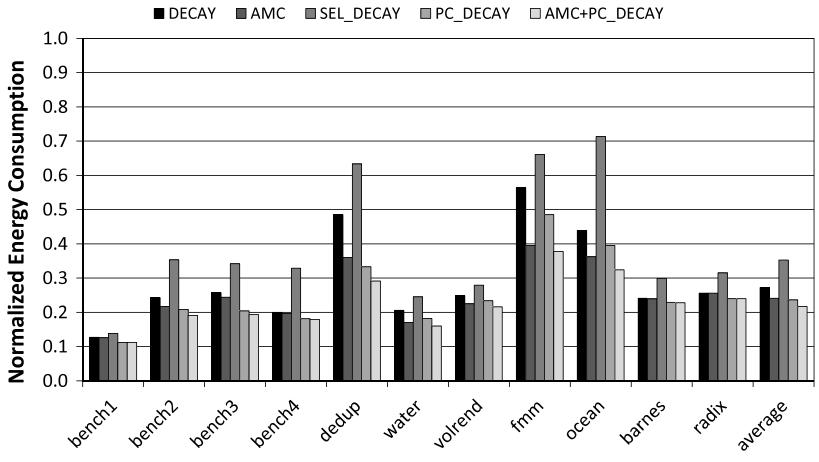
4.5. Sensitivity studies

We evaluate the energy consumption of each technique with different sizes of caches in order to show how the proposed technique affects energy consumption when various sizes of caches are used. Fig. 9(a) and (b) show the energy consumptions of benchmarks with 256 KB and 1 MB private L2 caches, respectively, while the energy consumption in 512 KB caches is shown in Fig. 6. As the sizes of caches become larger, the energy consumption reduced by the DECAY technique also increases because a larger number of cache blocks is in the idle state. In 256 KB private L2 caches, the DECAY technique reduces energy consumption by 38.8% on average while it reduces the energy consumption in 512 KB and 1 MB private L2 caches by 55.1% and 72.8%, respectively, compared to the baseline. With 256 KB caches, DECAY+LAPC and AMC+LAPC reduce energy consumption by 7.8% and 6.7% over DECAY and AMC, respectively. Similarly, with 1 MB caches, they reduce the energy consumption by 13.1% and 10.0% compared to DECAY and AMC as well. In particular, for dedup, DECAY + LAPC improves energy consumption by up to 31.3% over DECAY in 1 MB caches. This shows that, even in the various sizes of caches, the LAPC technique efficiently reduces energy consumption over the DECAY technique, and the AMC technique is also improved with it. The amount of energy reduction increases as the sizes of caches become smaller. In small caches, cache blocks are frequently replaced and thus increasing the number of N_{on} cache blocks in the Eq. (1). It increases the amount of energy consumption reduced by LAPC because LAPC can turn off the cache blocks which cannot be turned off in DECAY and AMC.

We also evaluate the proposed technique by varying the sizes of shared buffers, 16 KB, 32 KB, and 64 KB, as shown in Fig. 10. We show the normalized energy consumption with the different sizes of caches as well. In 256 KB caches, the DECAY + LAPC technique reduces energy consumption by 8.2%, 12.3%, and 14.3%



(a) 256 KB cache.



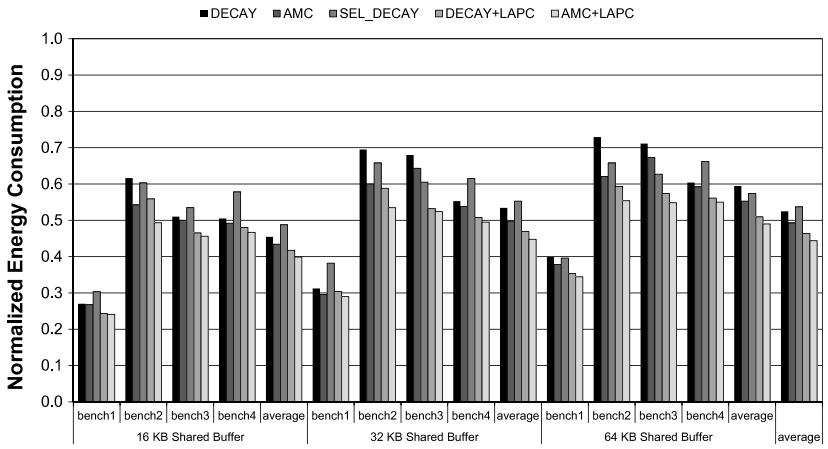
(b) 1 MB cache.

Fig. 9. Normalized energy consumption varying the size of caches.

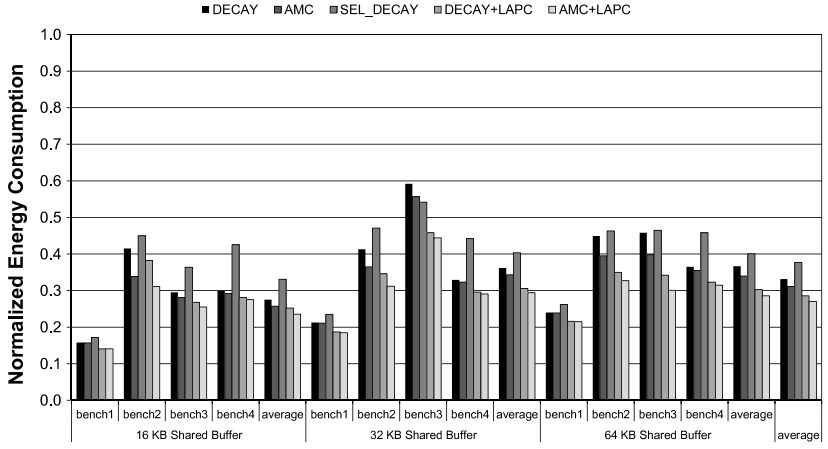
over the DECAY technique on average when shared buffers with the sizes of 16 KB, 32 KB, and 64 KB are employed, respectively. With these sizes of shared buffers, it reduces energy consumption by 8.4%, 15.4%, and 17.5% over the DECAY technique, respectively, in 512 KB caches on average. The reduction in energy consumption shows similar patterns even when the sizes of shared buffers become larger. However, the amount of energy reduction increases because of the increased ratio of the number of shared buffer cache blocks to that of other cache blocks. Furthermore, when the sizes of shared buffers are small, a producer thread writes its data to the same addresses of shared buffers every time it produces the data, and thus the shared buffer cache blocks are frequently accessed. This means that the corresponding cache blocks cannot be turned off when their time-out threshold values are high in DECAY and AMC. However, LAPC can turn them off by detecting the cache blocks which belong to shared buffers. In this case, LAPC can reduce energy consumption more by increasing N_{on} in the Eq. (1). On the other hand, when the sizes of shared buffers become larger, the data produced by a producer thread is written to the different addresses of shared buffers. Consequently, this increases the interval time between the accesses by a producer and a consumer to shared buffers. These cache blocks can be turned off in DECAY and AMC, but they should wait for the time-out threshold cycles, i.e., T_{DECAY} in the Eq. (1), to be turned off. The LAPC technique can turn them off earlier without waiting for T_{DECAY} cycles. In this case, shared buffer cache blocks can be turned off after waiting for only T_{LAPC} cycles, thereby increasing $E_{reduction}$ of the Eq. (1).

Since the performance of the LAPC technique is affected by the size of the global table employed for detecting the addresses, we evaluated the LAPC technique by varying its size. The evaluations are also done with different sizes of shared buffers because their sizes affect the detection ratio of the global table. As the sizes of shared buffers increase, the number of "ReadX" and "Read" transactions increases. If the number of entries in a global table is not large enough, only part of the shared buffer addresses can be detected. Fig. 11 shows the energy consumption with the different number of entries in the global table. LAPC_256, LAPC_512, LAPC_1024, and LAPC_2048 indicate the LAPC techniques with 256, 512, 1024, and 2048 entries in the global table, respectively, and all of the energy consumptions are normalized to LAPC_256. Fig. 11(a)–(c) show the energy consumptions when the 16 KB, 32 KB, and 64 KB shared buffers are used, and Fig. 11 (d) shows that of PARSEC and SPLASH2 parallel benchmarks.

As can be seen, by using the global table with less than 1024 entries, the reduction in energy consumption is smaller than that of the global table with more than 1024 entries because the global table is so small that the entry allocated by a ReadX transaction is evicted before a Read transaction of the corresponding address occurs on a shared bus. On the other hand, we can see that the global table more than with 1024 entries do not provide more advantages, which means the global table with 1024 entries is enough to detect shared buffers with less than 64 KB. Although all of the addresses are not detected in the global table with smaller entries, the consecutive addresses of the shared buffers can be detected because they have a high level of spatial locality.



(a) 256 KB cache.



(b) 512 KB cache.

Fig. 10. Normalized energy consumption varying the size of shared buffers.

4.6. Migration of shared buffer cache blocks

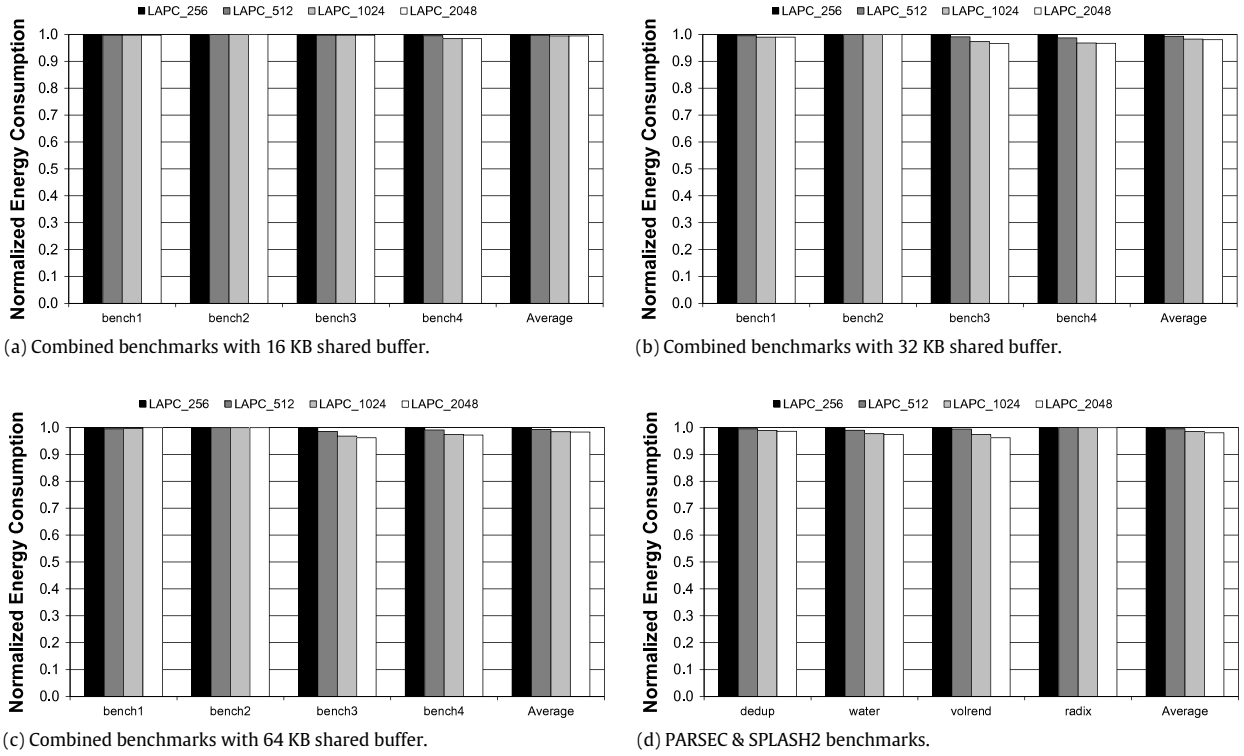
The proposed LAPC technique consists of two parts: (1) detecting shared buffer address space and (2) applying an aggressive decay technique to detected cache blocks. However, after detecting shared buffer address space, the corresponding cache blocks could be directly migrated to a consumer's cache to reduce the number of cache accesses and bus transactions instead of turning off the detected shared buffer cache blocks earlier. We also evaluate this option by comparing it with the DECAY and the LAPC techniques. Since the goal of this paper is reducing power consumption, the cache decay technique is basically applied to all cache blocks in both the LAPC and the LAPC_MIG techniques.

In the original private L2 cache organization, when a producer thread writes data to its shared buffer, a cache access to its local L2 cache occurs along with a ReadX bus transaction. Subsequently, when a consumer thread reads the data from the shared buffer, two cache accesses such as the local and the remote ones, a Read transaction, and a bus access to transfer the data occur. On the other hand, if the detected cache blocks are directly migrated to a consumer's L2 cache while not storing them in the local L1 and L2 caches, the remote L2 cache access and the ReadX and the Read bus transactions are not necessary because a consumer thread does not need to access a producer's cache to bring cache blocks. However, the bus and the cache accesses are still needed in that the former one is to migrate the data and the later one is to write the produced data to a consumer's cache. The energy consumption of the cache accesses and the bus transactions reduced by migrating the cache blocks is much smaller than the leakage energy consumption of a cache block as shown in Table 2. The leakage energy consumption of a cache block is 98 uW while the dynamic energy consumption of a cache access is 0.12 nJ and that of a bus transaction is 0.006 nJ. Therefore, the energy reduction by reducing the number of these accesses is not much larger than the energy reduction by turning off cache blocks.

Moreover, when the detected cache blocks are directly inserted into a consumer's cache, they could turn on the cache blocks which are already turned off or evict the cache blocks which are in use, thereby increasing energy consumption as well as decreasing performance. Fig. 12 shows the energy consumptions of LAPC and LAPC_MIG normalized to DECAY. LAPC_MIG is the technique that directly migrates the detected cache blocks to a consumer's cache. When compared to DECAY and LAPC, LAPC_MIG increases energy consumption by 3.4% and 16.0% on average, respectively. This is because the migrated cache blocks turn on the already turned-off cache blocks despite the smaller energy reduction by the removed cache access and bus transactions. The LAPC technique also slightly increases the execution time when the cache blocks are migrated too early. They could be evicted before being used or a consumer's cache blocks in use are replaced with them. As a result, when cache block turn-off techniques are considered for reducing leakage energy consumption, the proposed technique is a better choice.

5. Conclusions and future work

Reducing leakage energy consumption of the on-chip L2 cache has become an important issue in designing modern CMPs



(a) Combined benchmarks with 16 KB shared buffer.

(b) Combined benchmarks with 32 KB shared buffer.

(c) Combined benchmarks with 64 KB shared buffer.

(d) PARSEC & SPLASH2 benchmarks.

Fig. 11. Normalized energy consumption varying the size of a global table.

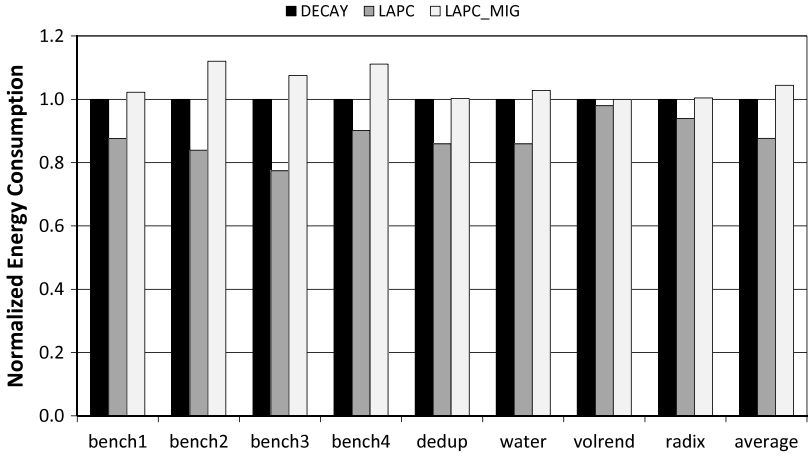


Fig. 12. Normalized energy consumption of LAPC_MIG.

because most CMPs employ a large on-chip L2 cache to improve performance. Although the existing cache leakage management technique can reduce the leakage energy consumption, we showed that exploiting knowledge about the behavior of caches, producer–consumer sharing in this case, can significantly increase the reduction in leakage energy. LAPC detects shared buffer addresses and aggressively turns off L2 cache blocks belonging to the detected shared buffers. Experimental results showed that the proposed technique can reduce the energy consumption in the L2 caches and off-chip memory by up to 31.3% compared with the existing cache decay technique without any loss in performance.

Our proposed technique can be extended in three ways. First, LAPC can be implemented to support a larger number of processors and more complex interconnections with directory-based cache configuration although, in this version of work, LAPC is developed based on 4 processors connected to a shared bus where a snoop-based coherence protocol is used. Second, the proposed leakage management technique can exploit characteristics of multi-threaded applications with other kind of data sharing patterns such as data parallel applications. Finally, a new leakage management technique for a shared L2 cache can be developed also by exploiting the characteristics of multi-threaded applications.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education, Science and Technology (MEST) (No. 20110020426, No. 20110020514, No. R33-2011-10095). The ICT at Seoul National University and IDEC provided research facilities for this study.

References

[1] K. Banerjee, A. Mehrotra, A power-optimal repeater insertion methodology for global interconnects in nanometer designs, IEEE Transactions on Electron Devices 49 (11) (2002) 2001–2007.

[2] N. Barrow-Williams, C. Fensch, S. Moore, A communication characterisation of splash-2 and parsec, in: Proceedings of the IEEE International Symposium on Workload Characterization, 2009, pp. 86–97.
[3] B.M. Beckmann, M.R. Marty, D.A. Wood, Asr: adaptive selective replication for CMP caches, in: Proceedings of the International Symposium on Microarchitecture, 2006, pp. 443–454.
[4] K.-S.S.J.P. Bienia, C., K. Li, The parsec benchmark suite: characterization and architectural implications, in: Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2008, pp. 72–81.
[5] Cacti 6.5, in: http://www.hpl.hp.com/research/cacti/.
[6] Calculating memory system power for ddr, in: Micron Technology Inc., 2005.
[7] J. Chandarlapati, M. Chaudhuri, Lemap: controlling leakage in large chip-multiprocessor caches via profile-guided virtual address translation, in: Proceedings of the International Conference on Computer Design, 2007, pp. 423–430.
[8] J. Chang, G.S. Sohi, Cooperative caching for chip multiprocessors, in: Proceedings of the International Symposium on Computer Architecture, 2006, pp. 357–368.
[9] L. Cheng, J.B. Carter, D. Dai, An adaptive cache coherence protocol optimized for producer–consumer sharing, in: Proceedings of the International Symposium on High Performance Computer Architecture, 2007, pp. 328–339.
[10] D.E. Culler, J.P. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann, 1999.
[11] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, T. Mudge, Drowsy caches: simple techniques for reducing leakage power, in: Proceedings of the International Symposium on Computer Architecture, 2002, pp. 148–157.
[12] M. Ghosh, H.S. Lee, Virtual exclusion: an architectural approach to reducing leakage energy in caches for multiprocessor systems, in: Proceedings of the International Conference on Parallel and Distributed Systems, 2007, pp. 1–8.
[13] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, Mibench: a free, commercially representative embedded benchmark suite, in: Proceedings of the International Workshop on Workload Characterization, 2001, pp. 3–14.
[14] ITRS (International Technology Roadmap for Semiconductor), in: http://public.itrs.net.
[15] S. Kaxiras, Z. Hu, M. Martonosi, Cache decay: exploiting generational behavior to reduce cache leakage power, in: Proceedings of the International Symposium on Computer Architecture, 2001, pp. 240–251.
[16] H. Kim, J.H. Ahn, J. Kim, Replication-aware leakage management in chip multiprocessors with private l2 cache, in: Proceedings of the International Symposium on Low Power Electronics and Design, 2010, pp. 135–140.
[17] D. Kim, S. Ha, R. Gupta, Cats: cycle accurate transaction-driven simulation with multiple processor simulators, in: Proceedings of the Design, Automation and Test in Europe, 2007, pp. 749–754.
[18] C. Lee, M. Potkonjak, W.H. Mangione-Smith, Mediabench: a tool for evaluating and synthesizing multimedia and communications systems, in: Proceedings of the International Symposium on Microarchitecture, 1997, pp. 330–335.
[19] S. Manne, A. Klauser, D. Grunwald, Pipeline gating: speculation control for energy reduction, in: Proceedings of the International Symposium on Computer Architecture, 1998, pp. 132–141.
[20] J.W. McPherson, Reliability challenges for 45 nm and beyond, in: Proceedings of the Design Automation Conference, 2006, pp. 176–181.
[21] M. Monchiero, R. Canal, A. González, Power/performance/thermal design-space exploration for multicore architectures, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 666–681.
[22] M. Monchiero, R. Canal, A. González, Using coherence information and decay techniques to optimize l2 cache leakage in CMPs, in: Proceedings of the International Conference on Parallel Processing, 2009, pp. 1–8.
[23] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, T.N. Vijaykumar, Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories, in: Proceedings of the International Symposium on Low Power Electronics and Design, 2000, pp. 90–95.
[24] M.K. Qureshi, Adaptive spill-receive for robust high-performance caching in CMPs, in: Proceedings of the International Symposium on High Performance Computer Architecture, 2009, pp. 45–54.
[25] V. Salapura, M. Blumrich, A. Gara, Improving the accuracy of snoop filtering using stream registers, in: Proceedings of the Workshop on Memory Performance: Dealing with Applications, Systems and Architecture, 2007, pp. 25–43.
[26] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, A. Gupta, The splash-2 programs: characterization and methodological considerations, in: Proceedings of the International Symposium on Computer Architecture, 1995, pp. 24–36.
[27] C. Yu, P. Petrov, Latency and bandwidth efficient communication through system customization for embedded multiprocessors, in: Proceedings of the Design Automation Conference, 2008, pp. 766–771.
[28] H. Zhou, M.C. Toburen, E. Rotenberg, T.M. Conte, Adaptive mode control: a static-power-efficient cache design, Transactions on Embedded Computing Systems 2 (3) (2003) 347–372.

Hyunhee Kim received her B.S. degree in computer science and engineering from the Chungang University, Seoul, Korea, in 2004, and her M.S. degree in computer science and engineering from Seoul National University, Korea, in 2006. She is currently working toward a Ph.D. degree at Seoul National University. Her research interests include high-performance and low-power chip multiprocessor architecture and on-chip memory management.

Jihong Kim received his B.S. degree in computer science and statistics from Seoul National University, Seoul, Korea, in 1986, and his M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, WA, in 1988 and 1995, respectively. Before joining SNU in 1997, he was a Member of Technical Staff in the DSPS R&D Center of Texas Instruments in Dallas, Texas. He is currently a Professor in the School of Computer Science and Engineering, Seoul National University. His research interests include embedded software, low-power systems, computer architecture, and multimedia and real-time systems.