Exploiting Replicated Cache Blocks to Reduce L2 Cache Leakage in CMPs

Hyunhee Kim, Jung Ho Ahn, *Member, IEEE*, and Jihong Kim, *Member, IEEE*

***Abstract*— Modern chip multiprocessors (CMPs) employ large L2 caches to reduce the performance gap between processors and off-chip memory. However, as the size of an L2 cache increases, its leakage power consumption also becomes a major contributor to the total power dissipation. Managing the leakage power of L2 caches, therefore, is an important issue in realizing low-power CMPs. In CMPs with private L2 caches, each processor makes a copy of the data in its local cache in order to access the data faster, which is called replication. In this paper, we propose a novel leakage management technique that dynamically turns off replications in private L2 caches for leakage power reduction by exploiting two key observations: 1) the cost of an extra cache miss due to the turned-off replication is small because the same cache block exists in another on-chip cache and 2) turning off the replication incurs no extra cache miss if it is invalidated by other processors in order to maintain cache coherence. Since blindly turning off the frequently accessed replications can degrade performance, the proposed technique dynamically controls the number of turned-off replications. The proposed technique can be implemented by slightly modifying the MESI protocol with a new turned-off shared (TOS) coherence state. The TOS state indicates that the corresponding block is shared by other caches but turned off. Experiments on a four-processor CMP with private L2 caches show that the proposed technique reduces the energy consumption of the L2 caches and the main memory by 19.4% on average, with less than 1% performance loss over the existing cache leakage management technique.**

***Index Terms*— Cache coherence, chip multiprocessors (CMPs), leakage power management, private L2 caches, replication.**

I. Introduction

RECENTLY, high-end mobile devices, such as laptops, smart phones, netbooks, and tablet PCs have become popular in everyday life. Since these systems are battery-operated systems, they should have low-power consumption although they also require powerful functionalities. In order to meet these requirements, chip multiprocessors (CMPs) are rapidly emerging as an alternative architecture for high-end embedded systems, providing high performance and low-power consumption. For example, ARM and Intel produce Cortex A9 MPCore Processor [1] and ATOM Processor [2], respectively. AMD also provides Turion Neo X2 and Athlon Neo X2 Dual-Core Processors for embedded applications [3]. The performance and power consumption of several high-performance energy-efficient multicore processors are also shown in [4]. In particular, Cortex A9 MPCore provides the peak performance of 4000 Dhrystone MIPS within 250 mW per CPU.

For these battery-operated systems, one of the most important issues is power dissipation, which consists of dynamic and static power. Dynamic power is dissipated due to the transistor switching activity whereas static power is mainly caused by sub-threshold and gate-oxide leakage. As the process technology advances below 65 nm, the leakage power becomes a major source of total power dissipation, which means that managing leakage power consumption is one of the critical design goals in realizing low-power CMPs [5]. Although the emerging technology, such as high-K dielectrics [6] has been introduced to reduce gate-oxide leakage, subthreshold leakage is still dominant in the total static power consumption [7].

On the other hand, since an on-chip L2 cache often determines the performance of CMPs, the current CMPs dedicate a large portion of their on-chip area to an L2 cache, making it a major power contributor. Even in the processors for mobile devices, an L2 cache becomes larger in order to meet their high-performance requirements. For example, most recently, Cortex A15 MPCore supports up to 4 MB of an L2 cache memory [1]. In these systems, since cores are less aggressively designed, thus their power dissipation may be 20–50 times smaller than those of servers, the caches can account for 25%–50% of the total power dissipation [8]. Fig. 1 shows a fraction of leakage and dynamic power consumption of cores, a shared bus, and private L2 caches, which is obtained when nine benchmarks from SPLASH2 [9] are executed using a multiprocessor simulator [10]. Four in-order cores, each of which has a private 512-KB L2 cache, are used. We estimated parameters for the power consumption of cores and caches from McPAT [11] and a shared bus from [12]. The experimental result shows that the leakage power consumption of L2 caches becomes almost 40% of the total power of cores, interconnect, and L2 caches.

A large body of previous research also shows that the leakage power consumption represents a significant portion of the total power consumption. For a single processor, it is known that 60% of total power dissipation in StrongARM

Manuscript received August 30, 2011; revised June 17, 2012; accepted August 31, 2012. Date of publication November 16, 2012; date of current version September 9, 2013. This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (MEST) under Grant 2012-0006417, the World Class University (WCU) Program through the NRF funded by the Ministry of Education, Science, and Technology under Grant R33-2012-000-10095-0, the ICT at Seoul National University, and IDEC. The work of J. Ahn was supported by the Research Settlement Fund for the new faculty of SNU.

H. Kim is with Samsung Electronics Co., Ltd., Suwon-si, 443-742, Korea (e-mail: hyunhee.kim@samsung.com).

J. Ahn is with the Graduate School of Convergence Science and Technology, Seoul National University, Seoul 151-742, Korea (e-mail: gajh@snu.ac.kr).

J. Kim is with the Department of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea (e-mail: jihong@davinci.snu.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2012.2220791

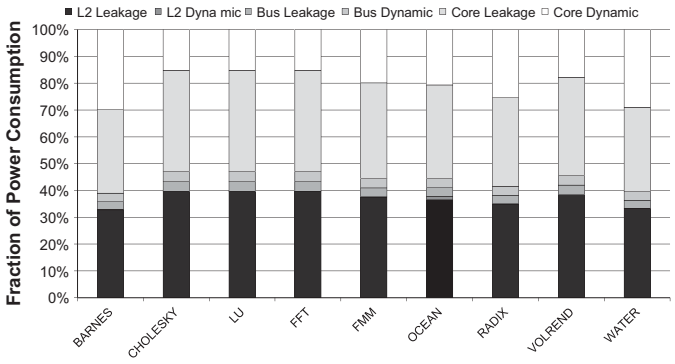

Fig. 1. Fraction of the dynamic and leakage energy consumption.

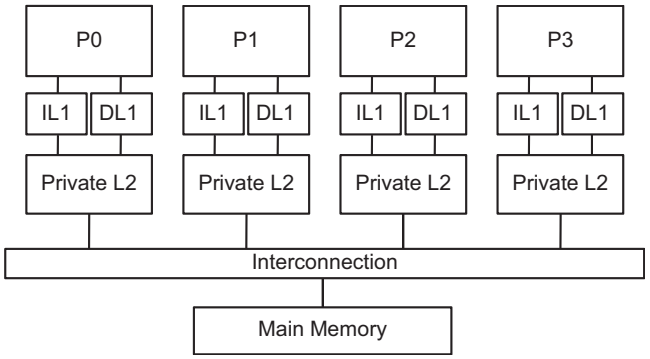

Fig. 2. Target architecture of CMPs.

is consumed by caches and memories [13]. For a multicore processor, an L2 cache in the Niagara-2 processor consumes more than 20% of total power dissipation while the power consumption of cores accounts for 30%. It is also shown that an L2 cache consumes 37% of total power on average when running parallel applications and 97% of this is the leakage power consumption [14]. Reference [15] has shown that leakage energy consumption ranges from 80% (two-core two-issue 8-MB L2) to 30% (eight-core eight-issue 1-MB L2) and most of it is consumed by L2 caches. Although a fraction of the power dissipated by an L2 cache is different depending on the architecture, these results emphasize that managing the leakage power consumption of an L2 cache becomes particularly important for modern CMPs.

There has been a large body of work focusing on reducing the leakage power consumption of a cache memory for both of the single processor [16]–[20] and CMPs [7], [14], [21], [22]. These techniques reduce the leakage power consumption by gating off a SRAM cell of inactive cache blocks as introduced in [19]. In single processor systems, a cache block is turned off by only considering the characteristics of an executing program and its own cache blocks. In [16]–[18], and [20], the cache block is turned off if it is not accessed for the predefined or dynamically adapted threshold time-out cycles, assuming that it is not accessed again. In the context of CMPs, many challenges present since more than one program or thread are simultaneously executed on multiple processors so that their cache blocks might interact each other. The works in [7], [21], and [22] exploit these newly introduced characteristics of cache blocks in CMPs, such as the multilevel inclusion property, cache coherence, and cache partitioning in order to save the leakage power consumption in L2 caches. However, these techniques do not exploit the sharing characteristics of cache blocks in multithreaded benchmarks.

In this paper, we propose architectural techniques to take advantage of a characteristic of multithreaded applications. In multithreaded applications, part of memory blocks is shared by multiple processors. To access the cache blocks faster, each processor makes the copy in its local cache, which is called a replication. Since this replication has its copy in another on-chip cache, turning it off has the benefit that the cost of an extra miss requires only an access to another on-chip private L2 cache rather than the off-chip access. Furthermore, a replication is often not likely to be accessed because it is replicated by another cache and then invalidated when its copy is updated. These characteristics allow us to reduce the leakage power without any significant performance loss by turning off a cache block immediately after another cache replicates it.

Based on these observations, this paper proposes a replication-aware leakage management (RALM) technique for private L2 caches of CMPs, which dynamically turns off a replication by exploiting its access characteristics. Fig. 2 shows the target architecture of the proposed technique, where each core has instruction and data L1 caches and a private L2 cache. Private L2 caches have shorter access latency than shared L2 caches because they are typically smaller than shared L2 caches. Furthermore, each processor makes replications of the shared data in its local private L2 cache in order to achieve the lower access latency by placing it close to the requesting processor.[1] These replications can make private L2 caches less capacity-efficient than shared L2 caches. However, prior research [23]–[25] has shown that private L2 caches achieve better performance if replications are efficiently managed. The private L2 cache organization is also more power efficient [24], [25] considering the following reasons: 1) a private L2 cache can be used as a unit for resource management to save energy when its processor is idle; 2) it can keep low set-associativity that consumes lower power; and 3) 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.

The proposed technique can be integrated by slightly modifying the existing MESI cache coherence protocol without increasing implementation cost. Unlike the original MESI cache coherence protocol, our modified cache coherence protocol has one new state, turned-off-shared (TOS), in which the cache block is shared by other processors but turned off. In some cases, if the TOS blocks are accessed frequently, the performance could be degraded since the data are brought from the remote L2 cache every time they are requested. The amount of performance degradation due to TOS blocks varies depending on the behavior of a program

[1]Even though current shared L2 caches are implemented with several banks to reduce the access latency, data might be located far away from the requesting core, thus causing latency to access the requested data. This becomes a more serious problem to solve as the number of cores increases.

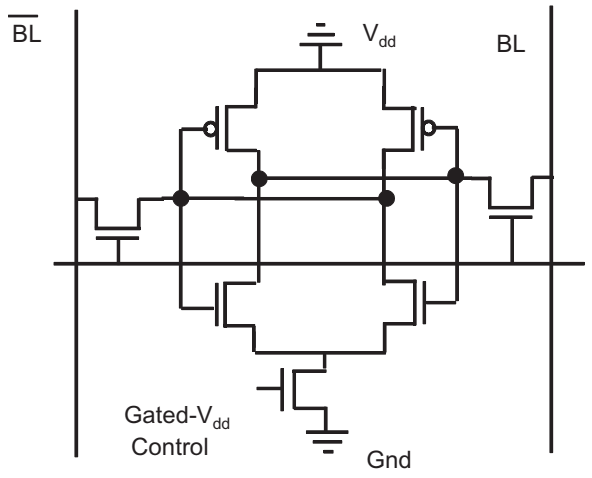

Fig. 3. SRAM cell with a gated-V_{dd}.

or its phase. The proposed RALM technique tracks performance degradation caused by TOS blocks for each period and dynamically regulates them in order to not degrade the performance below the predefined performance degradation threshold.

Experimental results show that the proposed technique reduces the energy consumption by 19.4% on average over the existing leakage management technique without significant performance degradation. The rest of this paper is organized as follows. In Sections II and III, we introduce the related work and motivation of our approach, respectively. Then, we describe the proposed RALM technique in Section IV. Experimental results are discussed in Section V, and conclusions are presented in Section VI.

II. Related Work

To reduce the leakage power consumption of a cache memory, there have been several efforts, such as [7], [14], and [16]–[22]. In these techniques, the leakage power consumption can be saved by gating off a SRAM cell as proposed in [19] and shown in Fig. 3. A cache decay technique [18] turns off cache blocks if they have not been accessed for time-out threshold cycles by predicting that they are not likely to be accessed in the near future. If the prediction is correct, the leakage power consumption can be saved during the turned-off period without any performance loss. However, it causes extra off-chip accesses when the turned-off cache blocks are accessed again since gating off a SRAM cell does not preserve the data. On the other hand, the drowsy cache [17] supplies the minimum power to preserve data to overcome the drawbacks of the cache decay technique. Although its performance degradation is less than the cache decay technique, the leakage power reduction also decreases. Adaptive mode control (AMC) [20] proposes to dynamically change the time-out threshold value during runtime and thus uses the smaller value if the overall performance does not degrade. This allows the cache blocks to be turned off earlier than when the static predefined threshold value is used. The cache decay technique is also developed for L2 caches to save the large leakage power consumption for single processor systems [16]. It proposes a smart predictor to determine an adaptive threshold value for each cache block based on the access interval between hits.

For multiprocessor systems, virtual exclusion [21] exploits one of the multiprocessor characteristics, multilevel inclusion. It turns off the repetitive cache blocks in the L2 caches if L1 caches have the data, but it is applicable only when the sizes of the L1 and L2 cache blocks be the same. Monchiero *et al.* [7] also reduced the leakage power consumption of the L2 caches in CMPs by minimizing the possibility that dirty blocks are turned off in order to improve the performance of decay. It also proposes the technique that turns off cache blocks when they are invalidated, which can reduce the leakage power consumption without any performance loss. Reference [22] proposes a power-aware cache partitioning technique that combines power-gating and cache partitioning, which are important design issues in designing high-performance and low-power shared caches for CMPs. LEMap [14] also aims at reducing leakage power consumption in last level cache by a novel virtual address translation technique, which requires modification in OS.

Although the existing leakage management techniques for CMPs are efficient, they can be improved by exploiting the characteristics of replications in CMPs. In terms of the performance improvement, several previous techniques propose to dynamically allocate replications in the private L2 cache organization of CMPs [23], [24] in order to balance between capacity and latency. Cooperative caching [24] replicates cache blocks with a given probability varying from 0% to 100%. ASR [23] limits the number of replications if the costs of allocating replications exceed its benefit. These techniques control the number of replicated cache blocks statically or dynamically, and the cache space for the replicated cache blocks is used for new data in order to reduce the cache hit rate. These are complementary to the proposed technique. On the other hand, [26], [27] introduce self-invalidation in order to reduce cache coherence overhead caused by replications. However, in these techniques, the energy consumption of the L2 caches is not considered, which is an important factor in designing low-power CMPs.

Sharing patterns of replications are also considered in order to improve the performance of a cache and its coherence protocol. Prior research [28]–[31] classifies the shared cache blocks into several particular patterns. [28] classifies the shared data into a small number of categories and shows that how an adaptive caching mechanism can achieve the performance improvement. Reference [32] also classifies the shared cache block into three groups: 1) read_only; 2) migratory; and 3) producer_consumer. Reference [29] and [31] adapt the cache coherence protocol to migratory sharing, which occurs when shared data is modified in a critical section. These techniques do not consider energy consumption of an on-chip cache memory. In addition, [30] classifies the access patterns into produce_consumer, migratory, multiple read/write, and multiple reader, and the proposed coherence protocol reduces the average L1 miss penalty by using adaptive replication, migration, and producer–consumer optimization. This technique only considers the dynamic energy consumption.

III. Motivation

The proposed technique is based on a simple observation that many of the replications can be turned off to save the leakage power consumption without negatively affecting the overall system performance. In this section, we first classify the access patterns of the replications into four major groups similar to prior research [29], [31], and analyze their characteristics in order to exploit them in reducing the leakage power consumption of a private L2 cache.

Typically, in multithreaded benchmarks, a memory block is referenced by only one processor, it is called an exclusive block while the memory block is called a shared block when it is referenced by more than one processor. When each processor has its local cache, the shared block is replicated in the requesting processor's cache, which is a replication. In this paper, we classify the sharing patterns of the replications into read_only, migratory, producer_consumer, and multiple_rw. Replications are classified as read_only sharing if they are written zero or one time and then read by another processor which does not write to them. This sharing pattern can be observed either when data is input to a program and other threads read them or when program codes are shared by multiple threads. Migratory sharing is observed when several threads read and write their shared data within a critical section, and it occurs commonly in multithreaded benchmarks. Cache blocks with this sharing pattern are replicated and then invalidated by another processor. Producer_consumer sharing occurs when processors write to and read from particular memory blocks in turn. Typically, a producer thread writes the produced data into the shared buffer memory space and a consumer thread reads the data written by the producer thread from it. If replications are read and written by multiple processors but do not have any particular patterns, we classify them as multiple_rw.

In this paper, unlike the previous research, we analyze and exploit the sharing characteristics of replications in reducing the leakage energy consumption. Especially, the proposed technique focuses on read_only and migratory sharing patterns. The prior research [33] has also proposed a leakage management technique for CMPs with private L2 caches while it targets cache blocks with producer_consumer sharing patterns. This technique can be easily integrated into the proposed technique because it is complementary to the proposed technique. Our proposed technique aims to dynamically turn off read_only and migratory replications in order to reduce the leakage power consumption of L2 caches based on the following observations.

1) In the target architecture, on a cache miss, data can be loaded into its local L2 cache from another processor's L2 cache (if it has the requested data) that has a shorter access latency than the off-chip memory. This allows that replications can be turned off aggressively to save the leakage power without decreasing the overall performance if it is guaranteed that their replications exist on-chip. Although the turned-off replications are needed again, they can be brought from another processor's cache.

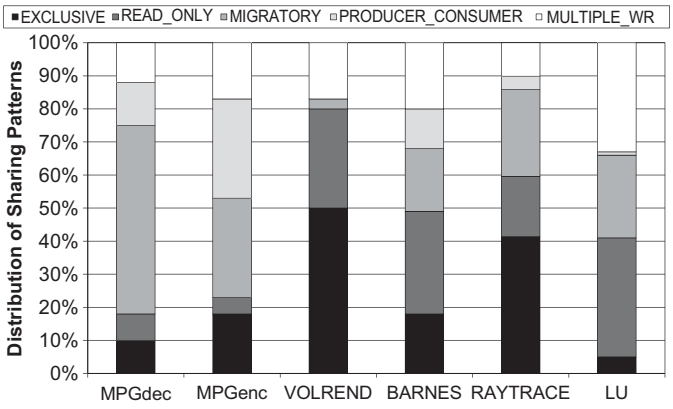

Fig. 4. Distribution of replications.

2) For replications with the migratory sharing pattern, they are invalidated by other processors after they are replicated and their copies in the replicating processors' caches are modified. This allows us to turn them off before they are invalidated without any performance loss.
3) For replications with the read_only sharing pattern, they can be turned off without an expensive cost while their copy is on-chip cache. Moreover, if they are accessed only once or a few times after they are allocated, turning them off can reduce the leakage power consumption without affecting the performance. However, unlike the migratory replications, it could cause the performance degradation if they are accessed frequently. Although these extra misses only require accesses to another processor's on-chip cache, the performance can be degraded significantly. In this context, it is necessary that these replications are dynamically turned off if they might incur performance drop.

Fig. 4 shows the distribution of sharing patterns among the allocated cache blocks in L2 caches for MPGdec and MPGenc of ALPBench [34] and VOLREND, BARNES, RAYTRACE, and LU of SPLASH2 [9]. The sharing patterns of the cache blocks are identified by keeping track of their access history while they are in the cache. Once a cache block is invalidated after it is allocated in the cache, the next miss to that cache block allocates a new cache block. This indicates that the number of all allocated cache blocks is larger than the number of all accessed memory blocks. A sufficiently large private L2 cache, which is a 16-way 16-MB cache for each processor, is used in order to contain the working set of most programs, thus avoiding capacity misses. As can be seen, more than 50% of the cache blocks allocated during the program execution are replications. In particular, for MPGdec, 57.0% of the allocated cache blocks are classified as migratory replications while read_only replications account for 8.0%. For other benchmarks, there are also a large portion of read_only and migratory replications. This means that the proposed technique can save the leakage power consumption without significantly decreasing the overall performance by turning them off efficiently. Furthermore, the proposed technique can be easily integrated into the existing MESI cache coherence

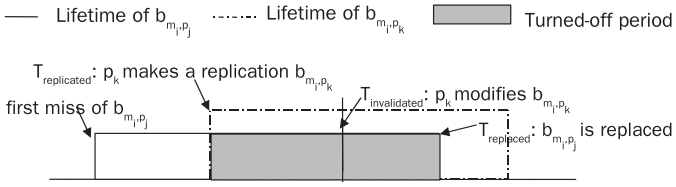

Fig. 5. Lifetime of the replications.

protocol with small modifications. Since all of the actions required to turn on/off the replications are performed when the transactions in the original MESI cache coherence protocol occur, the proposed technique can be implemented with a small hardware overhead.

IV. Leakage Management by Turning Off Replications

A. Cache Decay Technique

The proposed technique is based on the time-out based cache decay [18] technique. In this technique, a cache block is turned off if it is not accessed during the preset time-out cycles. In order to keep track of the elapsed cycles from the last access, two levels of counters, global and local counters, are used. The global cycle counter sends a tick signal to the local counters every certain cycles, and the local counter of each cache block is incremented by one whenever it gets the tick. When the local counter reaches the time-out threshold, the corresponding cache block is turned off. When the cache block is accessed, its local counter is reset to zero. In the experiment, we use four million cycles for the time-out threshold [18]. We also empirically observe that it is enough not to incur many extra misses in private L2 caches.

The proposed scheme is also based on the private L2 cache organization that uses the inclusion property [35], which is usually used in multilevel caches to efficiently implement a cache coherence in multiprocessor systems. In inclusive caches, the cache blocks that exist in the higher level caches should exist in the lower level caches. In this way, only the tags and states of the lower level caches are checked when cache coherence protocol messages are received by keeping the states of the blocks of the higher level caches in the lower level caches. Otherwise, both of the higher and lower level caches might check their tags and states every time, the transaction is presented on a snooping bus. This causes a significant performance degradation of the higher lever caches.

This inclusion property should also be considered even when employing the cache decay technique. If the tags are turned off with their data by the cache decay technique, the corresponding L1 cache blocks should be invalidated in order to meet the inclusion property, which can incur performance degradation. Therefore, we turn off only data portions of the cache blocks, keeping the tags and states of the blocks active. It allows only tags and states of the L2 caches to be responsible for maintaining the cache coherence as in a private L2 cache organization that does not employ the cache decay technique. Even though the proposed technique does not turn off the tags and states, it can significantly reduce leakage power because the energy consumption of the tags and states is relatively smaller than that of data blocks. When the dirty blocks in the L2 caches are turned off, the data should be written back to the memory. If the corresponding L1 cache blocks are in dirty states, they are also invalidated. In [7], it is shown that invalidating the dirty data can improve the system performance when the leakage management technique is applied to L2 caches. Since the time-out threshold value is long enough, the turned-off blocks of the L2 cache and the invalidated blocks of the L1 cache do not incur extra misses.

B. Algorithm of RALM

We assume that m memory addresses $m_1, m_2, \ldots, m_m$ are referenced by n processors $p_1, p_2, \ldots, p_n$ during the execution in parallel programs. In the parallel programs, a memory address $m_i \in \mathbf{M}$ can be referenced by only one processor or multiple processors. In this paper, a cache block for a memory address m_i, which is requested by a processor p_j, is denoted by b_{m_i,p_j}. When a processor p_j references a memory address m_i, only one cache block b_{m_i,p_j} exists in p_js on-chip L2 cache. On the other hand, when multiple processors, e.g., p_j and p_k, request the same memory address m_i, two cache blocks b_{m_i,p_j} and b_{m_i,p_k} are allocated in the L2 caches of p_j and p_k. In this case, if p_j requests m_i first, b_{m_i,p_j} is allocated in p_js L2 cache from the off-chip memory. After that, if p_k requests m_i, the replication of b_{m_i,p_j}, say b_{m_i,p_k}, is allocated in p_ks L2 cache. These cache blocks, b_{m_i,p_j} and b_{m_i,p_k}, become replications.

In order to describe the turned-off period of the replications when using the proposed RALM technique, Fig. 5 shows the lifetime of two replications of a memory address m_i, b_{m_i,p_j}, and b_{m_i,p_k}, which are allocated in the private L2 caches of the processors, p_j and p_k. The lifetime of b_{m_i,p_j} is overlapped by that of b_{m_i,p_k} during the time when b_{m_i,p_k} is replicated and b_{m_i,p_j} is replaced. For migratory replications, if p_k modifies its copy b_{m_i,p_k}, b_{m_i,p_j} is invalidated before it is replaced. Since b_{m_i,p_j} is not accessed during the period $T_{\text{invalidated}} - T_{\text{replicated}}$, it can be turned off immediately after being replicated without any performance loss. On the other hand, for read_only replications, b_{m_i,p_j} can also be turned off during the period $T_{\text{replaced}} - T_{\text{replicated}}$. As explained above, these read_only replications can be accessed during the turned-off period. If many of them are frequently accessed during this period, the performance may decrease. Our proposed technique also considers this performance degradation and dynamically turns off the replications in order to avoid it.

Algorithm 1 shows the algorithm of the proposed RALM technique, which includes operations that should be performed when read miss, read hit, and write hit occur in a processor p_k. On the cases that are not shown here, the cache operates in the way the same as the original private L2 cache that employs a MESI cache coherence protocol. When p_k reads the memory block m_i but the read miss occurs in its L2 cache, the memory block is brought from the L2 cache of another processor or the off-chip memory. In this context, if another processor p_j ($j \neq k$) already has a valid cache block b_{m_i,p_j}, it flushes its data to p_k while turning off b_{m_i,p_j}. It should be noted that if

Algorithm 1 Algorithm of RALM

```
1: if (read miss for m_i occurs in p_k) then
2:     if (another processor p_j has a valid b_{m_i,p_j}) then
3:         if (b_{m_i,p_j} is dirty) then
4:             write data back to the memory;
5:         end if
6:         turn off b_{m_i,p_j};
7:         C_{b_{m_i,p_j}} = 0;
8:     end if
9: else if (read hit for m_i occurs in p_k) then
10:    if (b_{m_i,p_k} is shared and turned off) then
11:        if (C_{b_{m_i,p_k}} < τ) then
12:            C_{b_{m_i,p_k}} += 1;
13:        else
14:            turn on b_{m_i,p_k};
15:        end if
16:    end if
17: else if (write hit for m_i occurs in p_k) then
18:    if (b_{m_i,p_k} is shared and turned off) then
19:        turn on b_{m_i,p_k};
20:        invalidate other copies of m_i on-chip;
21:    end if
22: end if
```

b_{m_i,p_j} is dirty, the data is written back to the off-chip memory before the cache block is turned off.

In order to avoid the performance degradation explained above, we employ an access counter for each cache block, e.g., $C_{b_{m_i,p_j}}$ for b_{m_i,p_j}, which keeps track of a number of accesses that occur after its cache block is turned off. This $C_{b_{m_i,p_j}}$ is reset to zero when the cache block is turned off. When p_k reads m_i but it has b_{m_i,p_k} that is turned off, the proposed technique only increments $C_{b_{m_i,p_k}}$ by one instead of turning on the cache block while fetching the data from another processor's cache (or from the off-chip memory if it does not exist) if its $C_{b_{m_i,p_k}}$ is less than a threshold τ. However, if $C_{b_{m_i,p_k}}$ reaches τ, the proposed technique turns on the corresponding cache block to avoid the performance degradation. On the other hand, when the replication is modified after being turned off, it should be turned on to keep the new data as its $C_{b_{m_i,p_k}}$ is reset to zero. In order to keep performance below the predefined performance degradation threshold, the RALM technique dynamically changes τ during runtime, which is described in Section IV-D.

C. Modified Cache Coherence Protocol

Since the actions in the RALM algorithm are activated by the transactions used in the cache coherence protocol, we integrate the proposed technique into the original cache coherence protocol with only a small modification to it, instead of implementing the RALM algorithm in separate hardware. Fig. 6(a) and (b) shows state transition diagrams for the original and modified MESI cache protocol, respectively.

The original MESI protocol consists of four states: 1) modified (M); 2) exclusive (E); 3) shared (S); and 4) invalid (I).

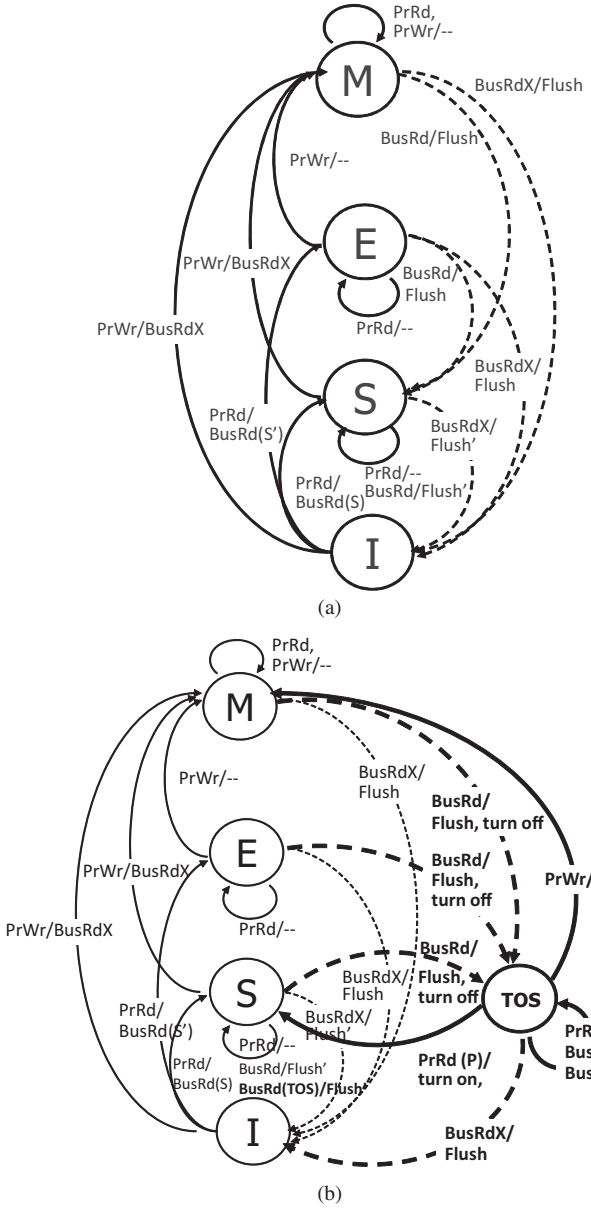

Fig. 6. Original and modified cache coherence protocols. Dashed and Solid lines represent additional transitions for RALM. TOS indicates an additional TOS state in which the corresponding cache block is shared by other caches but turned off. (a) Original cache coherence protocol. (b) Modified cache coherence protocol.

In the diagram, the notation "A/B" indicates that when the controller observes the transaction "A" from a processor or bus, it generates the bus transaction or action "B" while changing the state. "—" means that no action occurs. For both of the original and modified protocols, the solid arcs represent transitions due to local processor transactions while the dashed arcs represent transitions due to bus transactions. In the original MESI, when a cache block is first read by a local processor, PrRd in the diagram, it enters the S state if the copy of it exists in another cache. In this case, it also generates the BusRd transaction to make the other copies enter the S state. If its copy does not exist, it enters the E state. When the cache block in the E state is written by a local processor, PrWr, it can transition to the M state without generating a bus transaction

because no other cache has a copy while writing to the cache block in the S state generates the BusRdX transaction on the bus to invalidate the copies in other caches.

On the other hand, in the modified version, the TOS state is added to the original one. In this state, only the data block is turned off while its tag is kept turned on but treated the same as in the S state. The only difference from the S state is that when the cache access occurs, the data is brought from the other caches as the miss occurs. This additional state is needed in order to maintain the access counters for each L2 cache block. In the original protocol, the cache block enters the S state from M, E, or S when the BusRd transaction presents on the bus while supplying the data to the requesting cache. However, in the modified version, when the cache block at one of those states receives the BusRd transaction from the bus, its state is changed to the TOS state instead of the S state while the data block of it is turned off. In this case, the data is flushed to the requesting cache in the same way as the original one.

The cache block in the TOS state does not change its state when the PrRd transaction from the local processor is presented, which means that it remains continuously turned off but only its access counter is incremented by one. The requested data is fetched from another processor's cache by generating a BusRd transaction along with a TOS signal, which does not turn off the cache block. The TOS block is turned on only when either of these two following transactions occurs. First, when the PrWr transaction from the local processor occurs, the cache block is turned on while changing its state to M. Second, when the PrRd transaction occurs and the access counter of the TOS block reaches the threshold τ, the data block is turned on while changing its state to S because it is the frequently accessed block. Multiple copies can be in the S states if they are frequently accessed at the same time. Whether the access counter reaches τ or not is indicated as "P" in the diagram.

Fig. 7 shows the organization of RALM. The global counter sends a tick signal to the local counters (one per cache line) as in the cache decay technique. However, in RALM, power mode control (PMC) is added to turn off the cache block based on the local counter signal and the state transition. When the local counter reaches the threshold value or the state of the cache block is changed to TOS or I, PMC turns off the cache block by switching the supply voltage to 0 V. As in [7], RALM turns off a cache block when it is invalidated. When the cache block is accessed after being turned off, PMC switches its supply voltage to 1.0 V to turn it on. The additional circuit is not on the critical path because the state transition time is the same as in the original cache protocol and the comparison between the access counter and the τ can be processed when the tag matching occurs.

RALM can also be employed on CMPs with a directory-based cache coherence protocol. In this architecture, when the requested data is turned off, a request should be first sent to its home directory with the directory information in order to identify which core has a copy of the data. If the core's L2 cache with the home directory (called a home node) has a copy of the data, it sends the data to the requesting core. However,

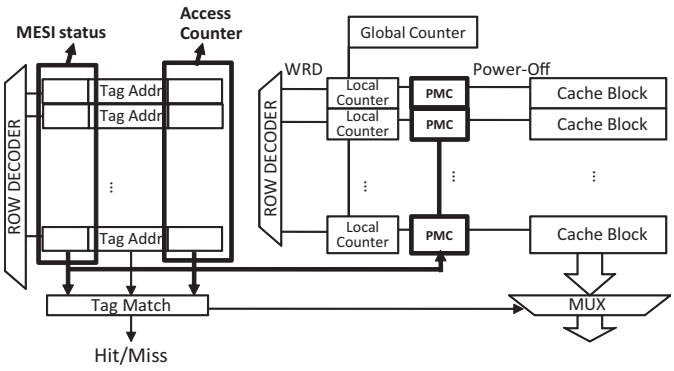

Fig. 7. RALM organization. Bold solid lines represent additional structures. PMC: power mode control.

if the home node does not have a copy, it selects the core which has a copy of the data and forwards the request to the selected core. Then, the core's L2 cache with the data sends them to the requesting core after receiving the forwarded message. ACK/NACK messages can be used to indicate whether the data is successfully forwarded to the original requesting core or not. Only after the home directory receives ACK or NACK messages, the data or their directory state can be updated.

D. Dynamic Control of TOS Blocks

In RALM, as shown in Section IV-B, a cache block dynamically enters a TOS state if the value of its access counter is lower than a threshold τ. However, the best value of τ for each program and each program phase is different depending on how many times and how frequently the TOS blocks are accessed during an execution. In some applications with only a few replications that are not frequently accessed, the performance does not degrade even when the high threshold value is employed. On the other hand, the performance could be deteriorated with the high threshold value when many blocks enter the TOS state and they are frequently accessed after being turned off. For these applications, the number of TOS blocks should be restricted to minimize the performance degradation. The proposed technique dynamically controls the number of TOS blocks by changing a threshold τ depending on the behavior of the program and its phase during the runtime.

Algorithm 2 shows how the RALM technique dynamically determines τ. RALM estimates the performance degradation ratio caused by TOS blocks during the runtime and keeps it below the maximum performance degradation ratio, MaxPerfDegradationRatio, by adjusting τ. On every predefined number of instructions, which is two million instructions in the evaluation, RALM calculates the amount of performance degradation due to the TOS blocks, denoted as PerfDegradationOfRALM. It can be estimated by the ratio of the additional cycles due to the TOS accesses, AdditionalCyclesWithRALM, to the execution cycles, TotalCycles, during the period. AdditionalCyclesWithRALM is calculated by multiplying the number of the accesses to the TOS blocks, NumAccessesToTOS, to the average remote cache access latency, AvgRemoteLatency.

Algorithm 2 Dynamic Control Algorithm of RALM

```
1: AdditionalCyclesWithRALM = NumAccessesToTOS *
     AvgRemoteLatency;
2: PerfDegradationOfRALM = AdditionalCyclesWithRALM
     / TotalCycles;
3: if (PerfDegradationOfRALM > MAXPerfDegradationRa-
     tio) then
4:     τ /= 2;
5:     if (τ == 0) then
6:         τ = MinTau;
7:     end if
8: else
9:     τ *= 2;
10:    if (τ > MaxTau) then
11:        τ = MaxTau;
12:    end if
13: end if
```

In this paper, we use the static value for AverageRemoteLatency. After PerfDegradationOfRALM is calculated, it is used to adapt the threshold value, τ. If PerfDegradationOfRALM is higher than MaxPerfDegradationRatio, RALM decreases τ by the half of it. On the other hand, if performance degradation is less than MaxPerfDegradationRatio, which means the TOS blocks are not accessed frequently, τ is doubled.

The proposed dynamic control mechanism is simple, and this simplicity is one of the advantages of the proposed technique in terms of implementation. Since τ is determined periodically, the proposed technique achieves a great energy reduction without any performance loss if the behavior of the workload does not significantly change. In this case, the determined τ works well for the next periods. For the workload which oscillates between different phases, the value of τ determined in the previous period might not be exactly suitable in the next period. This makes the proposed technique less effective. However, the proposed dynamic control mechanism also does not significantly degrade the overall performance because τ is only doubled or halved through phases, which means that it is not drastically changed according to the behavior of the workload.

Once τ is determined, it is applied to the entire cache blocks even though the numbers of accesses to each cache block are different from one another. In this technique, every cache block whose access counter is larger than τ does not enter the TOS state. However, this dynamic threshold adjustment technique can prohibit a cache block with a small number of accesses from being turned off and thus decreases energy reduction of the RALM technique. This result indicates that τ should be dynamically changed based on the prediction of the number of accesses to each cache block. In Section V, we evaluate energy reduction when only the cache blocks with a large number of accesses are disallowed to enter the TOS state. The number of accesses is predicted based on the access history of each memory address. We obtained this by storing the number of accesses of all memory addresses during runtime. The evaluation results show that the proposed RALM technique, which uses the same value τ for all cache blocks, can reduce energy consumption more aggressively even though its performance degradation is slightly larger than the block-level dynamic control. Moreover, this block-level dynamic control requires more complex implementation and a large area overhead to predict the number of accesses to each cache block. The proposed dynamic control technique can efficiently reduce energy consumption only with a small hardware overhead.

The TOS state can increase the number of remote private cache accesses, increasing network-on-chip power consumption. This means that if the number of the remote accesses significantly increases, the increased energy consumption can outnumber the energy saving achieved by TOS blocks. However, since we mainly focused on cache blocks with migratory and read-only sharing patterns, remote private cache accesses are not common. The cache blocks with migratory patterns are not likely to be reused as described in Section III, which means that they do not increase the number of network-on-chip accesses after being turned off. When the cache blocks with read-only sharing patterns are frequently reused, the proposed dynamic control mechanism can restrict the number of TOS blocks so as to not increase the network-on-chip power. In Section V, experimental results show that RALM can efficiently reduce the energy consumption of the memory systems that include the shared bus, private L2 caches, and DRAM by not significantly increasing the network on-chip power.

The proposed technique described in this paper assumes an inclusive private L2 cache. However, it can also be applied to CMPs with other L2 cache organizations, such as noninclusive or cooperative private caches [24] because they also have replications. Under these organizations, as in the inclusive private L2 cache organizations, a cache block enters the TOS states when it is replicated. When a cache miss occurs because the cache block is turned off, the requested data can also be brought from another core's L1 or L2 cache. In these organizations, the proposed technique may cause more significant performance degradation than the inclusive private L2 caches since a large portion of cache access can be served by the corresponding L1 caches in inclusive private L2 caches. However, the dynamic control mechanism proposed in this paper can restrict the number of TOS blocks when they incur performance degradation.

V. Experimental Results

A. Simulation Environment

To evaluate our technique, we modified the CATS [10] multiprocessor simulator to use a snoop-based MESI protocol for the cache coherency that supports cache-to-cache transfer of the cache block among private L2 caches. Table I shows the simulation parameters used in evaluations. We evaluated the proposed technique with multithreaded benchmarks from ALPBench [34] and SPLASH2 [9].

B. Evaluated Schemes

In this paper, first, we evaluate the proposed RALM technique which does not control the number of TOS blocks, called

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Processor	4 ARM Cores, 1 GHz in-order, 2-issue width bimodal predictor, no prefetcher
Private L1 I-cache and D-cache	16 KB, 32 B block 2-way set-associative, 1 cycle latency
Private L2 Cache	512 KB, 8-way 64 B block, 8 cycle latency
Cache coherency protocol	MESI
Interconnect	AMBA AHB bus, 64-bit width, 10 mm
Off-Chip Memory	512 MB DDR2, 300 cycle latency

TABLE II
ENERGY PARAMETERS

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
Shared bus	Dynamic energy	0.006 nJ
	Leakage power	33 mW
Off-chip memory	Read/write dynamic energy	2 nJ
	Standby power	50 mW

RALM_NSEL, by comparing the existing leakage management techniques, such as DECAY and AMC. RALM_NSEL turns off cache blocks whenever they are replicated by other caches and thus could cause performance degradation if the turned-off replications are frequently accessed. We evaluated RALM_NSEL by combining it with DECAY and AMC, called RALM_NSEL and AMC+RALM_NSEL since the RALM technique can be easily combined with the existing time-out based leakage management techniques. The evaluation of this RALM_NSEL technique shows that the RALM technique should dynamically turn-off replications.

DECAY and AMC employ the cache block turn-off techniques in [18] and [20], respectively. While DECAY uses the predefined time-out threshold value and applies it to all cache blocks, AMC adaptively changes the time-out threshold value at runtime. In each period, this technique keeps track of the number of ideal misses that occur when the cache decay technique is not applied and the number of sleep misses that are caused by the cache decay technique. Based on this information, when the number of sleep misses becomes much larger than that of ideal misses, the time-out threshold value is increased for the next period. On the other hand, it is decreased when the number of sleep misses becomes much smaller than that of ideal misses. AMC can reduce energy consumption more than DECAY if it can decrease a time-out threshold value when its performance degradation is not significantly large. In AMC+RALM_NSEL, the cache blocks except for replications are turned off based on the time-out threshold value which is dynamically changed as in the AMC technique.

The energy and performance of all of the evaluated techniques are normalized to the baseline technique, which does not employ a time-out based leakage management technique. However, even in the baseline technique, we turn off the cache blocks whenever they are invalidated by a cache coherence protocol as proposed in [7]. It can efficiently save the leakage energy consumption of a cache without any performance loss. Furthermore, in all of the techniques evaluated in this paper, cache blocks are turned off when they are invalidated.

We also evaluate the RALM technique that uses static threshold values from four to 256. The RALM techniques with these static threshold values four, 16, 64, and 256 are indicated by RALM_TH4, RALM_TH16, RALM_TH64, and RALM_TH256, respectively. Finally, the RALM technique with the dynamic control mechanism is evaluated, which is referred to as RALM. Through the evaluation of RALM with static threshold values, we can see which threshold value achieves the best energy consumption and performance for each benchmark and how the dynamic control mechanism of RALM can maximize the energy reduction while not degrading performance to below the predefined performance degradation ratio. In the evaluation, it is set to 5%. In addition, as described in Section IV-D, RALM_DB is also evaluated. RALM_DB determines whether a replication is turned off or not at the block level. In this technique, only the cache blocks with a large number of accesses are disallowed to enter the TOS state by predicting the number of accesses to each cache block.

C. Energy Model

We considered both the dynamic and leakage power consumption of an L2 cache, a shared bus, and off-chip memory. Dynamic energy consumption includes the dynamic energy consumption of the extra L2 cache and memory accesses caused by turning off cache blocks too early. We also take into account the dynamic and leakage power consumption overhead of additional counters in the proposed technique. The ten-bit global counter and the 12-bit local counters used in the cache decay technique. In addition, the proposed technique requires access counters for each cache block to control the number of TOS blocks and one additional bit per cache block for the new state in the modified MESI protocol, which increases by 1% of the energy consumption and area of the total cache in comparison with the baseline technique. The dynamic control mechanism also requires a storage overhead per core, which becomes 2 bytes for AdditionalCyclesWithRALM, 1 byte NumAccessesToTOS, 1 byte for PerfDegradationOfRALM, 3 bytes for TotalCycles, and 1 byte for MAXPerfDegradationRatio. Therefore, the total storage overhead for the dynamic control of τ is 9 bytes, which is less than 1% of a private L2 cache area. The extra logic for the Gated-V_{dd} technique [19] is also considered, which becomes less than 5% of the total cache leakage. Table II shows the energy parameters that are obtained from CACTI 6.5 [36] using the 45-nm technology and the off-chip memory energy estimation [37]. The power consumption of a shared bus is also obtained using the bus power model in [12] and the wire properties based on ITRS [38]. Since the

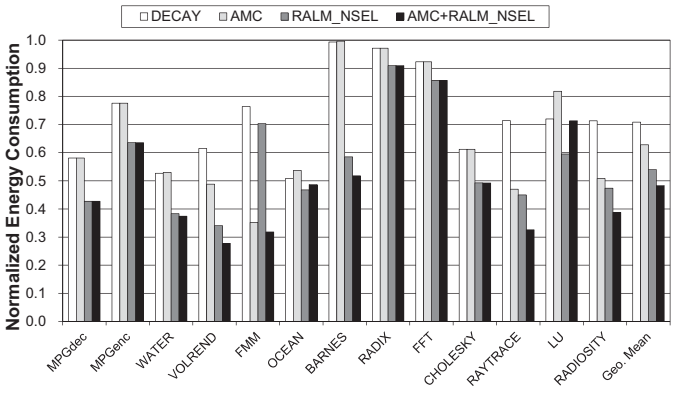

Fig. 8. Normalized energy consumption.

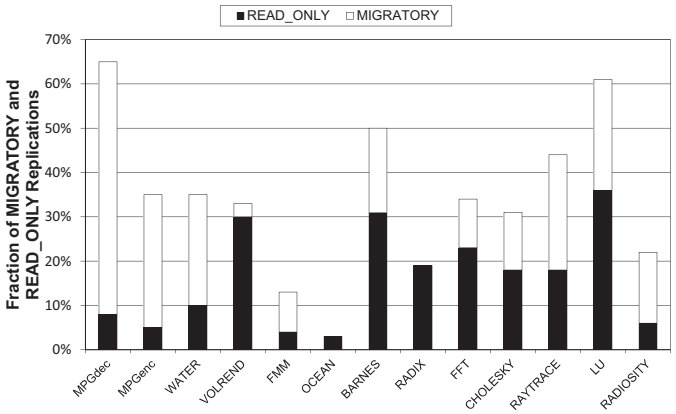

Fig. 9. Distribution of replications.

activity of a shared bus in the embedded systems is much lower than in the high-performance desktop computer, we assumed the activity factor α to be 0.15 in the power model. A private L2 cache can further reduce the activity of a shared bus compared to when the bus is placed between L1 D/I caches and a shared L2 cache because only misses to the private L2 cache access the shared bus [25].

D. Energy/Performance Evaluation

1) Evaluation of RALM With No TOS Control: Fig. 8 shows the normalized energy consumptions of DECAY, AMC, RALM_NSEL, and AMC+RALM_NSEL. RALM_NSEL reduces the energy consumption by 46.0% and 24.8% on average over the baseline technique and the DECAY technique, respectively. In particular, for MPGdec, MPGenc, WATER, BARNES, and VOLREND, RALM_NSEL can reduce the energy consumption by 26.5%, 18.0%, 27.3%, 41.1%, and 44.6% over DECAY, respectively, while RADIX, CHOLESKY, FFT, OCEAN show less than 10% energy reduction by turning off replications without waiting for the time-out threshold cycles. Meanwhile, the AMC technique reduces energy consumption on average by 11.4% compared to the DECAY technique, by reducing the time-out threshold value if the number of extra cache misses caused by the current time-out threshold value is not large. However, when the RALM_NSEL technique is combined with AMC, it can also reduce energy consumption by 31.8% and 23.0% over DECAY and AMC, respectively.

In the proposed technique, the amount of reduction in leakage power consumption depends on a fraction of replications with migratory and read_only sharing patterns. The cache blocks with migratory sharing patterns can be turned off after being replicated without any performance loss because the data are not needed any more. However, the energy reduction by migratory replications can be smaller than the energy reduction by read_only replications if the cache blocks with the migratory sharing patterns are invalidated in a short time after being replicated. On the other hand, although the energy reduction achieved by read_only replications can be larger than the energy reduction by the migratory replications, performance can be degraded by turning off the read_only frequently accessed replications.

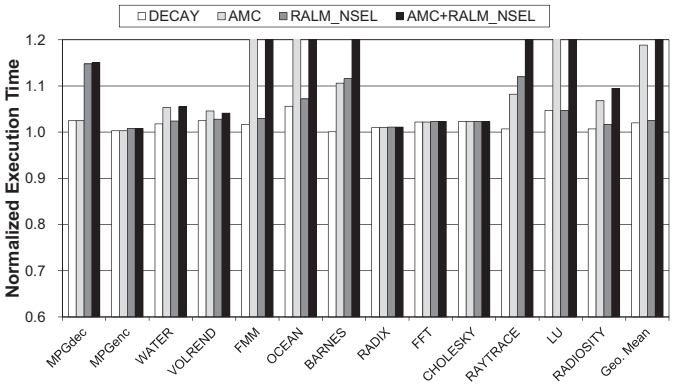

Fig. 10. Normalized execution time.

Fig. 9 shows how many replications with migratory and read_only sharing patterns are allocated during the execution of each benchmark. As can be seen, MPGdec, MPGenc, WATER, and LU have more than 20.0% of the migratory replications as analyzed in [32], thus showing significant energy reduction in RALM_NSEL with less than 2.0% performance loss compared to DECAY, as shown in Fig. 10. In particular, MPGdec has 57.0% of the migratory replications and thus RALM_NSEL reduces energy consumption by 26.5% over DECAY. In this benchmark, RALM_NSEL also decreases performance by 14.8% compared to the baseline technique because some of the turned-off read_only replications are frequently accessed. On the other hand, BARNES and VOLREND have a large portion of read_only, and thus RALM_NSEL can reduce energy consumption significantly. However, for BARNES, although RALM_NSEL reduces energy consumption by up to 41.1% over DECAY, it also degrades performance by 11.6%. It turns off a large number of read_only replications, but the most of the turned-off cache blocks are reused many times, thus degrading performance. This shows the necessity of the RALM technique that dynamically turns off replications, considering performance degradation.

On the other hand, we can see that the scientific benchmarks, such as RADIX, FMM, and FFT, and OCEAN from SPLASH2 do not have many migratory replications. In these benchmarks, RALM_NSEL has less opportunity to turn off

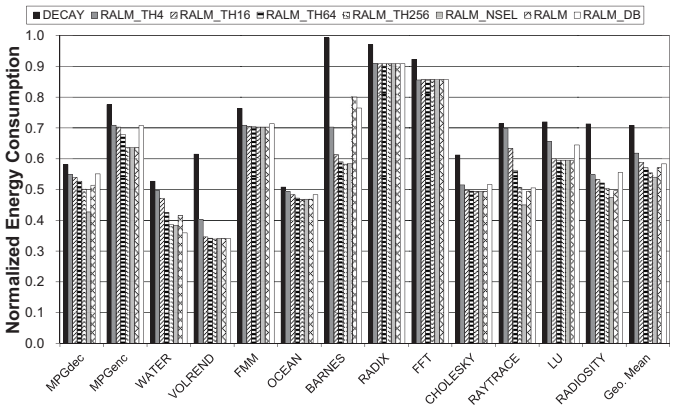

Fig. 11. Normalized energy consumption.

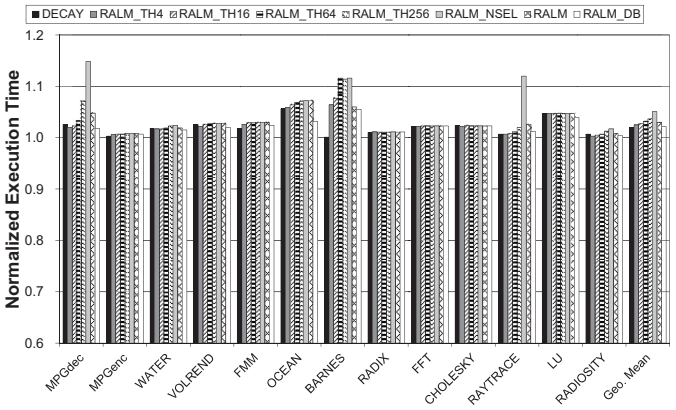

Fig. 12. Normalized execution time.

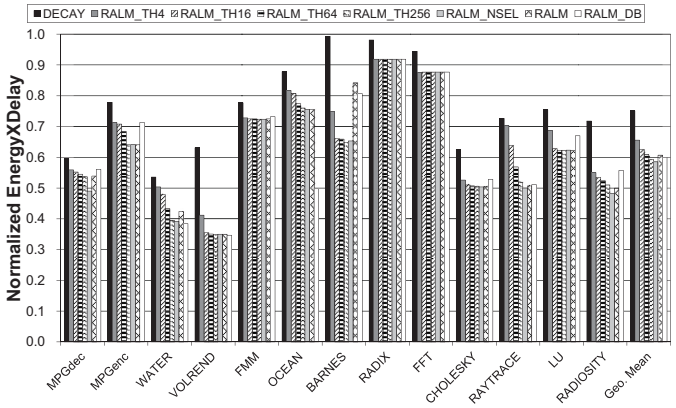

Fig. 13. Normalized energy delay product.

the cache blocks than other benchmarks. However, even in these benchmarks, RALM_NSEL reduces energy consumption by turning off read_only replications. RALM_NSEL does not degrade performance because the read_only cache blocks in these benchmarks are not accessed frequently after being turned off. Since the proposed technique does not invalidate the cache blocks in the L1 cache even when the corresponding L2 cache blocks are turned off, most of the read_only cache blocks can be hit in the L1 cache, thus not affecting the performance.

2) Evaluation of RALM With Dynamic Control: Fig. 11 shows the normalized energy consumption of RALM_TH4, RALM_TH16, RALM_TH64, RALM_TH256, RALM, and RALM_DB. As can be seen, for RALM with static threshold values, the amount of energy reduction increases as the threshold value becomes larger. For all of the benchmarks, RALM_TH256 shows the largest energy reduction because it turns off more replications than RALM_TH4, RALM_TH16, and RALM_TH64. As a result, RALM_TH4, RALM_TH16, RALM_TH64, and RALM_TH256 reduce energy consumption by 12.7%, 17.0%, 19.4%, and 21.8% on average, respectively, over DECAY by turning off the replications. Fig. 12 shows the normalized execution time of each technique. As can be seen, for MPGdec and BARNES, the performance of RALM_TH256 decreases by 7.1% and 11.6%, respectively, over the baseline technique while it shows the largest energy reduction. Although it can reduce performance degradation over RALM_NSEL by prohibiting the cache blocks which are accessed more than 256 times from being turned off, the performance degradation is still high. In particular, for BARNES, RALM_TH4 starts to degrade performance by 6.4%. This indicates that the RALM technique should adaptively decrease the threshold value less than four for BARNES.

RALM, which dynamically controls the threshold value, reduces energy consumption as much as RALM_NSEL for the benchmarks which do not show performance degradation. For these benchmarks, it increases the threshold value up to the maximum level because turning off cache blocks whenever they are replicated does not affect the overall performance. However, for the benchmarks which show a higher performance degradation ratio than the predefined maximum performance degradation ratio, RALM decreases the threshold value so that the performance degradation ratio is not higher than the predefined one. As a result, RALM reduces energy consumption by 42.9% and 19.4% on average with an average of less than 3.0% and 1.0% performance loss over the baseline technique and DECAY, respectively. This indicates that the cache blocks turned off by the RALM technique are not frequently reused, thus the network traffics do not significantly increase. Only for BARNES, RALM degrades the performance by up to 5.9%, which is caused by 17.0% increase in network traffics over DECAY, while keeping the performance for the other benchmarks by not increasing the network traffics. Although this slowdown can increase the leakage power consumption of interconnect and DRAM, the energy reduction achieved by the proposed technique can compensate for the increased energy consumption by turning off the large number of cache blocks. Fig. 13 shows the energy delay product of each technique normalized to the baseline technique. RALM can improve the energy delay product by 19.2% on average compared to DECAY, which is slightly larger than the best technique, RALM_TH256. This can be achieved by dynamically turning off replications while keeping performance.

The experimental results in this section show that the proposed technique can reduce the energy consumption depending on the fraction of the cache blocks with read-only and migratory sharing patterns. For example, RALM_NSEL and

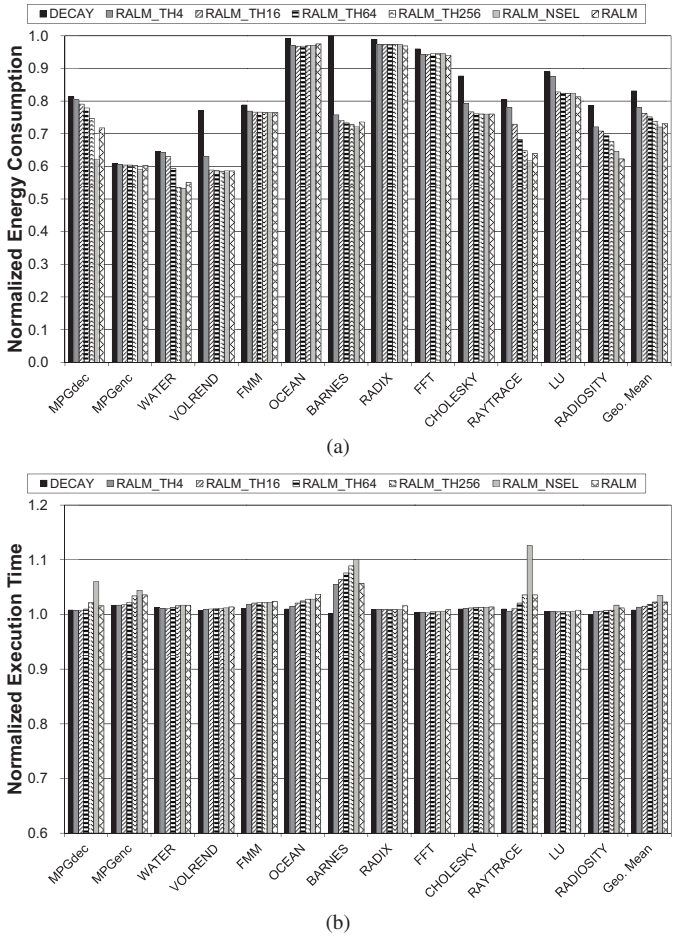

Fig. 14. Normalized energy consumption and execution time in a 256-KB cache. (a) Normalized energy consumption in a 256-KB cache. (b) Normalized execution time in a 256-KB cache.

RALM reduce the energy consumption by 29.2% and 23.6%, respectively, on average, over DECAY for benchmark programs with read-only and migratory sharing patterns among cache blocks. For example, MPGdec and MPGenc from ALPbench and WATER, VOLREND, BANRNES, CHOLESKY, RAYTRACE, LU, and RADIOSITY from SPLASH2 belong to this type of benchmark programs. This indicates that RALM can reduce the energy consumption if applications have a larger portion of cache blocks with read-only and migratory sharing patterns. As migratory sharing patterns are more readily adopted in CMPs [32], the RALM technique can reduce the energy consumption for other multiprocessor benchmarks as well. On the other hand, several applications, such as OCEAN, FMM, and RADIX, exhibit producer–consumer and multiple read–write patterns in addition to some degree of read-only and migratory sharing patterns. Since these applications do not have a high degree of read-only and migratory sharing patterns that RALM is targeting on, the amount of energy consumption reduced by the proposed technique is not so high as in the applications with a high degree of read-only and migratory sharing patterns. For example, for FMM and OCEAN, RALM reduces the energy consumption only by 8.0% and 7.9%, respectively. At the same time, FMM and OCEAN also experience a negligible performance loss, 1.3% and 1.5% only, respectively. For a cache block with the producer–consumer sharing patterns, such as in FMM and OCEAN, the producer core produces data in its local L2 cache and a consumer core makes a copy of them in its L2 cache from the producer's L2 cache. When the proposed technique is applied, the cache block in the producer core is turned off while entering the TOS status when the consumer core copies it. When the producer core modifies the replication, the turned-off replication is turned on while entering the M state as explained in Section IV-C. In this case, since the TOS block is turned on again immediately after being accessed, the performance does not significantly degrade. This subsequently decreases the amount of the power consumption reduced by RALM. This indicates that RALM does not negatively affect the power consumption and the overall system performance even when the applications do not exhibit the high degree of read-only and migratory sharing patterns.

RALM_DB shows that its energy consumption is larger than RALM by up to 11.9% while it can efficiently reduce performance degradation. This is because the number of accesses to each cache block depends on how frequently it is replaced during runtime. Although a memory block has been accessed a large number of times during runtime, its cache block can be accessed only a few times before the replacement. When this replication is not disallowed to enter the TOS state because the corresponding memory address is accessed frequently in the past, it loses a chance to be turned off, although it does not affect performance. In this context, it is necessary to predict how frequently each block is accessed and how much it degrades performance. However, it requires a complex prediction mechanism, whereas RALM can sufficiently reduce energy consumption with a small hardware overhead while keeping performance.

3) Sensitivity Studies:

a) Effect of cache size: Fig. 14(a) and (b) shows the normalized energy consumption and execution time, respectively, when using 256-KB private L2 caches. When using the 256-KB cache, RALM_TH4, RALM_TH16, RALM_64, RALM_256, RALM_NSEL, and RALM reduce energy consumption by 6.1%, 8.3%, 9.6%, 11.2%, 13.3%, and 12.0%, respectively, on average compared to DECAY. The amount of energy reduction decreases in a small cache. Cache blocks are frequently replaced in the small cache and thus decreasing turn-off time. However, similar to a 512-KB cache, the energy reduction increases as the threshold value becomes larger. Although FMM, OCEAN, RADIX, and FFT show the almost 2% of the energy reduction over DECAY, most of the benchmarks except for them show a larger energy reduction over DECAY. In particular, by using RALM_NSEL, a larger number of cache blocks with migratory sharing patterns in WATER, BARNES, RAYTRACE, and RADIOSITY, contribute significantly to reducing energy consumption by up to 17.5%, 24.0%, 27.9%, and 23.3%, respectively, over DECAY. In this cache size, RALM can also reduce energy consumption as well as keep performance by dynamically changing the threshold value for access counter. It reduces the threshold value as close to four for BARNES while keeping the threshold value as the maximum value for other benchmarks.

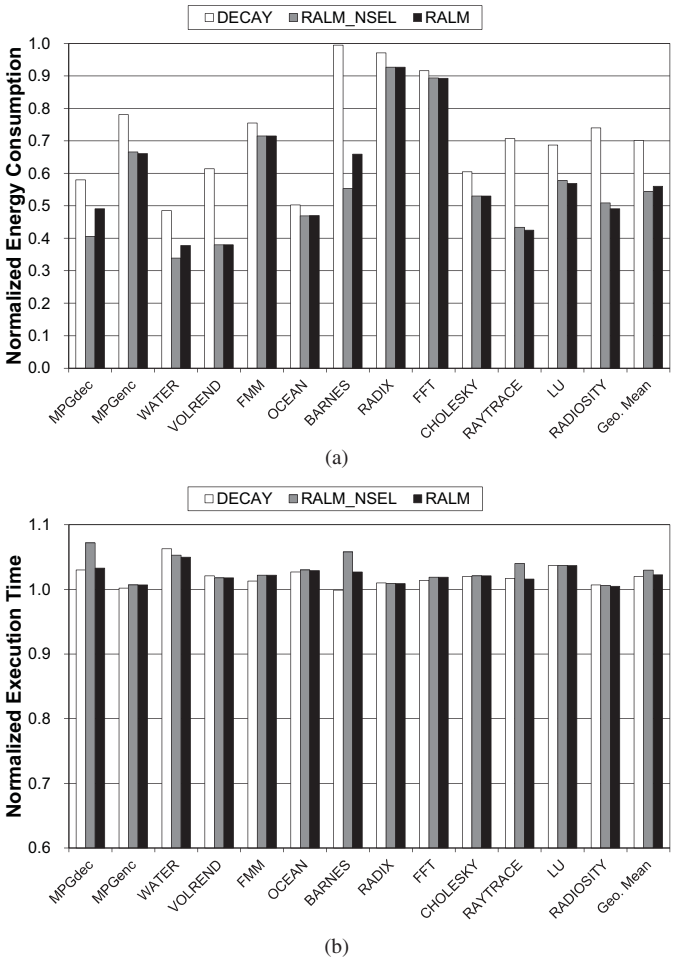

Fig. 15. Normalized energy consumption and execution time with a 32-KB L1 cache. (a) Normalized energy consumption. (b) Normalized execution time.

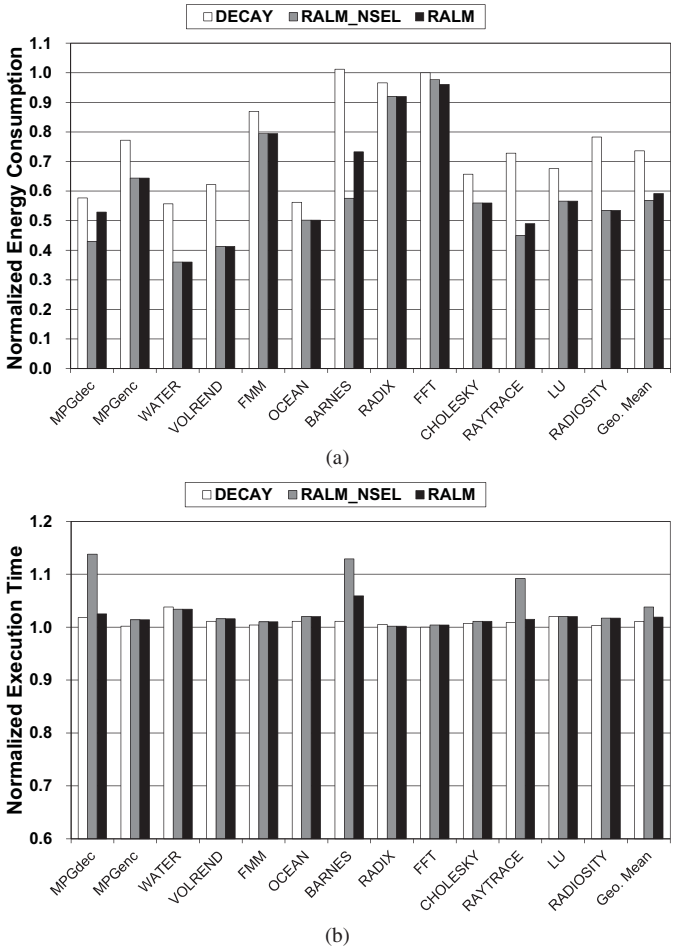

Fig. 16. Normalized energy consumption and execution time with 128-byte cache blocks. (a) Normalized energy consumption. (b) Normalized execution time.

The size of an L1 cache can affect the energy reduction and performance of the RALM technique because the data are kept in the L1 cache even when their corresponding cache blocks in a private L2 cache are turned off. Therefore, the larger L1 cache can reduce the performance degradation of RALM by reducing extra cache misses. Fig. 15(a) and (b) shows the normalized energy consumption and the execution time, respectively, when a 32-KB L1 cache is used. The performance degradation of RALM_NSEL is less than when a 16-KB L1 cache is used. RALM_NSEL reduces energy consumption by 45.6% and 44.0% on average, respectively, compared to the baseline and the DECAY techniques, while degrading the respective performances by 7.2% and 5.8% for MPGdec and BARNES, compared to the DECAY technique. For all benchmarks, RALM can reduce energy consumption by 20.0% while keeping the performance degradation ratio below 5.0%.

b) Effect of false sharing and prefetching: False sharing can affect the performance and energy reduction of the RALM technique. Since the proposed technique is applied to the unit of a cache block, the data which are not shared by multiple cores can be turned off if false sharing occurs. This can cause performance degradation if the data is reused after entering the TOS states. Especially, false sharing usually increases with the size of a cache block. We evaluated the RALM technique when larger cache blocks are used. The larger cache blocks can improve performance with high locality but incur false sharing which degrades the performance of RALM. Fig. 16(a) and (b) shows the normalized energy consumption and the execution time, respectively, when the size of a cache block is 128 bytes. As can be seen in the figures, the performance degradation of RALM_NSEL is by up to 13.8% and 12.9% for MPGdec and BARNES, respectively, which is larger than RALM_NSEL with 64-byte cache blocks. However, the RALM technique reduces energy consumption by 40.8% and 19.6% on average, compared to the baseline technique and the DECAY technique, respectively, while keeping the maximum performance degradation at 5.9% for BARNES. It can be achieved by dynamically turning off replications, depending on the behaviors of executed programs and their phases.

Prefetching can also reduce the amount of energy reduction when any leakage management technique is applied using the cache decay technique. That is because the cache blocks turned off by the cache decay technique should be turned on in order to store the prefetched data. Even though the total amount of energy reduction by turning off cache blocks decreases

when prefetching is employed, the proposed technique can also efficiently reduce energy consumption more than the original cache decay technique because it can turn them off earlier. However, a reasonable study on the effect of prefetching is out of the scope of this paper.

VI. Conclusion

We proposed a RALM technique, which is based on a power-efficient private L2 cache organization for CMPs. The proposed technique turns off replications immediately after another on-chip cache makes a copy of it. Turning off replications reduces leakage energy consumption without significant performance loss because the cost of an extra miss only requires an on-chip access, which is much faster than the off-chip access, and many replications were invalidated after making their copies in other caches. In order to avoid significant performance degradation caused by turning off frequently accessed replications, the proposed RALM technique dynamically controls the number of TOS cache blocks during the runtime. To turn off the replication efficiently without much hardware overhead, we slightly modified an original MESI cache coherence protocol. The experimental results showed that RALM reduces energy consumption by 42.9% and 19.4% compared to the baseline technique and the DECAY technique, respectively, on average without significant performance degradation.

In this paper, we only assumed the CMP architecture in which each processor was connected by a shared bus and cache coherence was maintained by a snoop-based MESI protocol. However, as the number of processors in CMPs increases, the shared bus with the snoop-based cache coherence protocol has a limitation in scalability. Therefore, in the future, we plan to propose a leakage management technique based on more complex interconnects with a directory-based cache coherence protocol.

References

[1] *Cortex-A Series*. (2012) [Online] Available: http://www.arm.com/products/processors/cortex-a

[2] *Intel Atom Processor*. (2012) [Online] Available: http://www.intel.com/products/processor/atom/index.htm

[3] *Introducing AMD Turion Neo X2 and AMD Athlon Neo X2 Dual-Core ASB1 (BGA)*. (2012) [Online]. Available: http://www.amd.com/us/Documents/47413A_Dual_Core_BGA_brief_PDF.pdf

[4] A. Munir, S. Ranka, and A. Gordon-Ross, "High-performance energy-efficient multicore embedded computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 4, pp. 684–700, Apr. 2012.

[5] N. Cheung. (2004). *EE143 General Information* [Online]. Available: http://public.itrs.net

[6] J. W. McPherson, "Reliability challenges for 45 nm and beyond," in *Proc. Design Autom. Conf.*, Jul. 2006, pp. 176–181.

[7] M. Monchiero, R. Canal, and A. González, "Using coherence information and decay techniques to optimize L2 cache leakage in CMPs," in *Proc. Int. Conf. Parallel Process.*, Sep. 2009, pp. 1–8.

[8] J. L. Hennessy and D. A. Patterson, *Computer Architecture, a Quantitative Approach*, 5th ed. San Mateo, CA: Morgan Kaufmann, 2011.

[9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Int. Symp. Comput. Arch. Conf.*, Jun. 1995, pp. 24–36.

[10] D. Kim, S. Ha, and R. Gupta, "CATS: Cycle accurate transaction-driven simulation with multiple processor simulators," in *Proc. Design, Autom. Test Eur.*, Apr. 2007, pp. 749–754.

[11] *McPAT*. (2008) [Online]. Available: http://www.hpl.hp.com/research/mcpat/

[12] K. Banerjee and A. Mehrotra, "A power-optimal repeater insertion methodology for global interconnects in nanometer designs," *IEEE Trans. Electron Devices*, vol. 49, no. 11, pp. 2001–2007, Nov. 2002.

[13] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: Speculation control for energy reduction," in *Proc. Int. Symp. Comput. Arch. Conf.*, Jul. 1998, pp. 132–141.

[14] J. Chandarlapati and M. Chaudhuri, "LEMap: Controlling leakage in large chip-multiprocessor caches via profile-guided virtual address translation," in *Proc. Int. Conf. Comput. Design*, Oct. 2007, pp. 423–430.

[15] M. Monchiero, R. Canal, and A. González, "Power/performance/thermal design-space exploration for multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 666–681, Jun. 2008.

[16] J. Abella, A. González, X. Vera, and M. O'Boyle, "IATAC: A smart predictor to turn-off L2 cache lines," *ACM Trans. Arch. Code Optim.*, vol. 2, no. 1, pp. 55–77, 2005.

[17] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in *Proc. Int. Symp. Comput. Arch.*, May 2002, pp. 148–157.

[18] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. Int. Symp. Comput. Arch. Conf.*, Jun. 2001, pp. 240–251.

[19] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proc. Int. Symp. Low Power Electron. Design Conf.*, May 2000, pp. 90–95.

[20] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive mode control: A static-power-efficient cache design," *ACM Trans. Embedd. Comput. Syst.*, vol. 2, no. 3, pp. 347–372, 2003.

[21] M. Ghosh and H. S. Lee, "Virtual exclusion: An architectural approach to reducing leakage energy in caches for multiprocessor systems," in *Proc. Int. Conf. Parallel Distrib. Syst.*, Dec. 2007, pp. 1–8.

[22] I. Kotera, K. Abe, R. Egawa, H. Takizawa, and H. Kobayashi, "Power-aware dynamic cache partitioning for CMPs," *Trans. High-Perform. Embed. Arch. Compil.*, vol. 3, no. 2, pp. 149–167, 2008.

[23] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: Adaptive selective replication for CMP caches," in *Proc. Int. Symp. Microarch. Conf.*, Dec. 2006, pp. 443–454.

[24] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *Proc. Int. Symp. Comput. Arch. Conf.*, Jun. 2006, pp. 357–368.

[25] M. K. Qureshi, "Adaptive spill-receive for robust high-performance caching in CMPs," in *Proc. Int. Symp. High Perform. Comput. Arch. Conf.*, Feb. 2009, pp. 45–54.

[26] A.-C. Lai and B. Falsafi, "Selective, accurate, and timely self-invalidation using last-touch prediction," in *Proc. Int. Symp. Comput. Arch.*, 2000, pp. 139–148.

[27] A. R. Lebeck and D. A. Wood, "Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors," in *Proc. Int. Symp. Comput. Arch. Conf.*, 1997, pp. 48–59.

[28] J. K. Bennett, J. B. Carter, and W. Zwaenepoel, "Adaptive software cache management for distributed shared memory architectures," in *Proc. Int. Symp. Comput. Arch. Conf.*, May 1990, pp. 125–134.

[29] A. L. Cox and R. J. Fowler, "Adaptive cache coherency for detecting migratory shared data," in *Proc. Int. Symp. Comput. Arch. Conf.*, May 1993, pp. 98–108.

[30] H. Hossain, H. Dwarkadas, and M. C. Huang, "Improving support for locality and fine-grain sharing in chip multiprocesors," in *Proc. Int. Conf. Parallel Arch. Compil. Tech.*, Oct. 2008, pp. 155–165.

[31] P. Stenström, M. Brorsson, and L. Sandberg, "An adaptive cache coherence protocol optimized for migratory sharing," in *Proc. Int. Symp. Comput. Arch. Conf.*, May 1993, pp. 109–118.

[32] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *Proc. IEEE Int. Symp. Workload Character. Conf.*, Oct. 2009, pp. 86–97.

[33] H. Kim and J. Kim, "A leakage-aware L2 cache management technique for producer-consumer sharing in low-power chip multiprocessors," *J. Parallel Distrib. Comput.*, vol. 71, no. 12, pp. 1545–1557, 2011.

[34] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," in *Proc. IEEE Int. Symp. Workload Charact.*, Oct. 2005, pp. 34–45.

[35] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. San Mateo, CA: Morgan Kaufmann, 1999.

[36] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. (2009). *McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures* [Online]. Available: http://www.hpl.hp.com/research/cacti/

[37] *Calculating Memory System Power for DDR*, Micron Technology Inc., Boise, ID, 2005.

[38] M. L. Mui, K. Banerjee, and A. Mehrotra, "A global interconnect optimization scheme for nanometer scale VLSI with implications for latency, bandwidth, and power dissipation," *Trans. High-Perform. Embedd. Arch. Compil.*, vol. 51, no. 2, pp. 195–203, 2004.

Hyunhee Kim received the B.S. degree from Chungang University, Seoul, Korea, in 2004, and the M.S. and Ph.D. degrees from Seoul National University, Seoul, in 2006 and 2011, respectively, all in computer science and engineering.

She is currently with the Mobile Communication Division, Samsung Electronics Co., Ltd. Her current research interests include embedded softwares, low-power systems, and computer architectures.

Jung Ho Ahn (M'07) received the B.S. degree from Seoul National University, Seoul, Korea, in 1997, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 2002 and 2007, respectively, all in electrical engineering.

He is currently an Assistant Professor with the Graduate School of Convergence Science and Technology, Seoul National University, where he leads the Scalable Computer Architecture Laboratory. He was a Research Scientist with Hewlett-Packard Laboratories, Palo Alto, CA, from 2007 to 2009. His current research interests include designing high-performance and high-efficiency memory systems and interconnection networks as well as bridging the gap between the performance demand of emerging applications and the performance potential of modern and future massively parallel systems.

Jihong Kim (M'00) received the B.S. degree in computer science and statistics from Seoul National University (SNU), Seoul, Korea, in 1986, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, in 1988 and 1995, respectively.

He was a Technical Staff Member with the DSPS R&D Center, Texas Instruments, Dallas. He joined SNU in 1997, where he is currently a Professor with the School of Computer Science and Engineering. His current research interests include embedded softwares, low-power systems, computer architectures, and multimedia and real-time systems.