A Leakage-Aware Cache Sharing Technique for Low-Power Chip Multi-processors (CMPs) with Private L2 Caches

Hyunhee Kim
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
hh0726@davinci.snu.ac.kr

Sungjun Youn
LG Electronics Corporation
Seoul 152-702, Korea
spica81@davinci.snu.ac.kr

Jihong Kim
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
jihong@davinci.snu.ac.kr

ABSTRACT

Power dissipation becomes an important issue in modern microprocessors such as chip multiprocessors (CMPs). Especially as the process technology advances below 90nm, the leakage power consumption becomes dominant in the total power dissipation, thus reducing the leakage power consumption is an important design goal for low-power CMPs. In particular, since most CMPs employ a large L2 cache, reducing the leakage power consumption of the L2 cache is critical in realizing low-power CMPs.

In this paper, we propose a leakage-aware on-chip L2 cache organization called LACS. The proposed LACS, like the existing RACS organization, is based on a private L2 cache organization with an inter-L2 cache sharing support. However, unlike the RACS organization, which determines a peer L2 cache block for an inter-L2 cache sharing based on the reusability of the evicted L2 block and performance implications of peer L2 cache blocks, the LACS organization considers both the performance and leakage. The LACS organization reduces the leakage power consumption significantly over the leakage-oblivious RACS organization while achieving a similar performance gain over a private L2 cache organization. Experimental results show that the proposed LACS technique reduces the energy consumption by 23.6% and improves the energy delay product by 18.6% on average over the existing RACS scheme.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Design Styles—*Cache memories*

General Terms

Design, Management

Keywords

Chip Multiprocessors, Leakage energy, Reusability, L2 cache management

1. INTRODUCTION

Power dissipation becomes a major concern to design modern microprocessors such as chip multiprocessors (CMPs). Especially, as CMOS technology advances below 90nm, the leakage power becomes the dominant source of power consumption, thus reducing the leakage power consumption is an important issue in realizing low-power CMPs. These CMPs generally have several levels of on-chip cache memories to hide the performance gap between the processors and the main memory. Since the on-chip cache memories often determine the performance of CMPs, many CMPs dedicate a large portion of their on-chip area for an L2 cache memory and modify a cache organization to use the on-chip cache space more efficiently [2, 3, 4, 12, 14, 15]. However, since a large L2 cache often dominates the on-chip leakage power consumption among all the on-chip components, reducing the leakage power consumption as well as improving the performance of the L2 cache for CMPs becomes more important.

There have been many research investigations on different on-chip L2 cache organizations that aim to improve the performance of a shared L2 cache or a private L2 cache for CMPs. CMP_SNUCA [2] scheme applies NUCA [8] to the CMP architecture, which migrates blocks close to a requestor to reduce wire delay. Similarly, Victim Replication [15] keeps L1 victims in a local L2 cache slice to reduce write delay in a shared L2 cache organization. CMPNuRAPID [4] makes copies close to requestors, which allows a fast access for read-only sharing. They propose capacity stealing of neighbor's cache when its capacity is not large enough to store private data. CMP_CC [3] writes an evicted block from a local cache to a peer L2 cache[1] randomly with a given probability from 0% to 100%. It allows to redistribute private L2 cache spaces by sharing their spaces. A cache sharing technique in a private L2 cache organization is also proposed in [12]. It selectively writes evicted blocks to peer L2 caches only when the peer cache has an invalid line or a shared line. On most existing on-chip L2 cache organizations, however, which support an inter-L2 cache sharing, the

[1]In this paper, a peer L2 cache indicates a private L2 cache of neighboring processor.

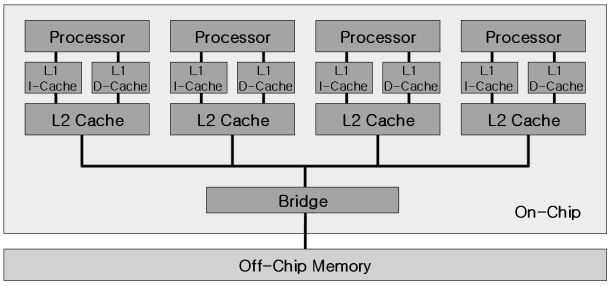

Figure 1: A target CMP architecture.

reusability of an evicted block is not considered. Considering the reusability of an evicted block is important because writing evicted blocks, which are not reused, generates additional on-chip shared bus transactions. These transactions may cause extra conflicts on a shared bus and pollute peer L2 caches.

RACS [14] is the first cache sharing technique that considers the reusability of evicted blocks based on a private L2 cache organization. In this scheme, when a block is evicted from a local private L2 cache, it is written to a peer L2 cache only if it is likely to be reused in the near future. This selective write back allows to reduce the shared bus traffic and avoid polluting the peer L2 cache. It predicts the reusability of evicted blocks by observing the Access Time Interval and Frequency (ATIF) patterns. Although RACS is effective in reducing the dynamic power consumption caused by writing evicted blocks to peer caches, however, it does not consider the leakage power consumption. In this paper, we extend the RACS scheme to better handle the leakage power consumption in the on-chip L2 cache organization.

To reduce the leakage power consumption of a cache memory, there have been several efforts such as [6, 7]. These proposals have reduced the leakage power consumption by turning off power supply to cache blocks. The cache decay technique [7] selectively turns off cache blocks which have not been accessed for a time-out threshold cycles. It, however, causes extra misses because the turned off cache blocks do not preserve data. On the other hand, the drowsy cache [6] preserves data to overcome the drawbacks of the cache decay technique. In order to avoid extra cache misses, it supplies the minimum power to keep the data. However, when these leakage reduction techniques were applied for a private L2 cache organization with an inter-L2 cache sharing support (such as CMP_CC and RACS), the leakage power consumption may increase over an organization which does not use the cache sharing technique. This is because writing evicted blocks to peer L2 caches may wake up turned off cache blocks or prevent a cache block from switching to a sleep state. Furthermore, if the blocks are not reused after they are written to peer L2 caches, the leakage power consumption can also increase.

In this paper, we propose a Leakage-Aware Cache Sharing technique called LACS which is based on a private L2 cache organization with an inter-L2 cache sharing. Figure 1 shows an overview of a target CMP architecture. Unlike the leakage-oblivious RACS technique, the proposed LACS technique improves both performance and energy consumption by selectively writing evicted blocks to peer caches considering both the reusability and the leakage energy savings. When a block is evicted from a local private L2 cache, the reusability of the block is checked to decide if it should be written to a peer cache or not in the same fashion as done in RACS [14]. However, unlike RACS, if it is likely to be reused in the near future, our scheme finds peer L2 cache blocks which are not likely to be turned off soon or are likely to be turned on in a short time. For this selection, we consider the blocks at the bottom of the LRU stacks of the peer L2 caches. If such blocks are available, LACS checks if there exists a block with low reusability among them. If the block has low reusability, it is replaced with the evicted block. Experimental results show that the proposed LACS technique reduces the energy consumption by 23.6% and improves the energy delay product by 18.6% on average over the existing RACS scheme while achieving a similar performance gain over a private L2 cache organization.

The rest of this paper organized as follows. In Section 2, we briefly review of the RACS scheme proposed in [14]. We explain the motivation of our approach and describe the details of the LACS technique in Section 3. Experimental results are discussed in Section 4, and we conclude the paper in Section 5.

2. REUSABILITY-AWARE CACHE SHARING TECHNIQUE

Several systems based on a private L2 cache organization transfer evicted blocks to peer L2 caches to use an L2 cache space efficiently. Since it takes longer to access the off-chip memory than the on-chip memory, these approaches can improve the performance. In CMP_CC [3], the L2 victim block from a local private L2 cache is written to a random peer cache with a given probability from 0% to 100%, but they do not consider the reusability of the evicted block. Speight *et al.* [12] propose selective writing between L2 caches, but they only write evicted blocks to peer L2 caches which have a shared line or an invalid line. However, it does not also identify which L2 victims are likely to be reused in the near future. If blocks are not likely to be needed again soon it is more desirable that they are not kept on chip because writing the evicted blocks to peer caches generates additional on-chip traffic and may evict peer L2 cache blocks that will be needed shortly. On the other hand, retaining blocks that will be reused can reduce an access latency by finding them in the on-chip L2 cache instead of in the off-chip memory. Therefore, the reusability of blocks should be considered to keep only the blocks which have high reusability on chip.

To decide the reusability of blocks, RACS classifies blocks by Access Time Interval and Frequency (ATIF) pattern which is based on the number of short time intervals and long time intervals to a block. It monitors the reuse ratio of the blocks for each pattern and write them to peer L2 caches when the corresponding pattern has high reuse ratio. Figure 2 shows what proportion of the blocks written to the peer L2 caches is reused or not for each ATIF pattern when all the evicted blocks are written to the peer caches. X-axis represents the 16 ATIF patterns. The first number of each pattern represents the number of short time intervals and the second number represents the number of long time intervals. In

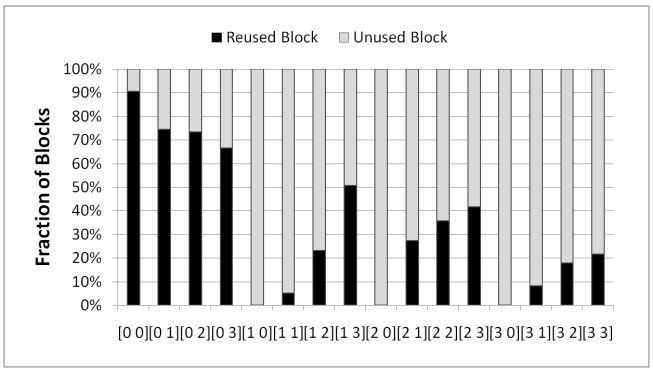

Figure 2: Distributions of reused blocks and unused blocks under different ATIF patterns [15].

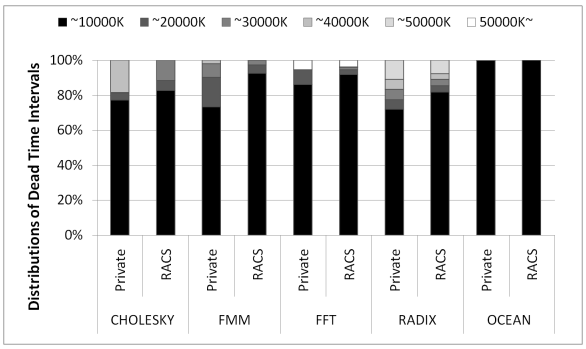

Figure 3: Distributions of dead time intervals.

most cases, ATIF patterns that correspond to a lot of accesses with a long time interval, such as [13],[22], [23], and [33], have the relatively larger number of reused blocks. But when the first number of the ATIF pattern, corresponding to the number of accesses after short time intervals, is zero, like [00], [01], [02], and [03], the blocks do not have a high temporal locality, and many of them are reused regardless of the second number of the pattern. Consequently, ATIF patterns used in RACS can identify blocks with a high reusability.

The RACS scheme also compares the memory demand of caches not to corrupt the L2 cache of processors with a high memory demand. If blocks with low reusability do not exist at the bottom of LRU stacks of peer caches, it checks if there are caches with a smaller memory demand and writes the evicted blocks to them. RACS predicts that a processor has a high memory demand as frequent replacements occur in a private L2 cache. The history of the time interval between subsequent replacements, $Repl_{history}$ is used as the prediction value of a processor's memory demand. Using the reusability and memory demand prediction techniques, the RACS scheme reduces the number of off-chip accesses by up to 17% over the pure private cache organization.

3. LEAKAGE-AWARE CACHE SHARING TECHNIQUE

3.1 Motivation

The RACS scheme described in Section 2 only considers the reusability of evicted L2 blocks and peer L2 cache blocks. It replaces the peer L2 cache block with the evicted block if the evicted block has a high reusability and the peer L2 cache block has a low reusability. However, writing the evicted block to the peer cache might decrease the leakage energy savings when the existing cache leakage management technique, such as cache decay [7], is applied to RACS. In the cache decay technique, the dead time interval of a cache block is defined as an interval between the time when the last hit occurs and the time when the next miss occurs in that cache block. A cache block could be turned off during the dead time interval except for initial time-out threshold cycles to reduce the leakage energy consumption. Therefore, a cache sharing technique may interfere with opportunities for leakage energy savings by reducing the length of dead time intervals. In particular, the following two cases may degrade the efficiency of the cache decay technique. First, if an evicted block is written to the turned off block in a peer L2 cache, the turned off block should be woken up to store the evicted block. Second, if an evicted block is written to the turned on block in a peer L2 cache which was not accessed for close to the threshold cycles, this cache block might lose a chance to be turned off because a time-out counter should restart from zero. Even though writing an evicted block improves the performance, RACS loses many chances for leakage energy savings under the cache decay scheme by preventing a cache block from entering a sleep state.

Figure 3 compares how dead time intervals are distributed between the private L2 scheme and the RACS scheme using the SPLASH2 benchmarks [13]. The private L2 scheme assumes a private L2 cache organization without an inter-L2 cache sharing support. When the RACS scheme is used, the number of long dead time intervals decreases over the private L2 scheme. For example, in CHOLESKY, about 18.5% of dead time intervals of the private L2 scheme have an interval length of about 40000K cycles. On the other hand, this percentage becomes 0% in the RACS scheme. Similarly, for FMM, the fraction of the dead time intervals longer than 20000K cycles decreases from 26.6% to 7.5%. With the decreased number of long dead-time intervals, the cache decay technique becomes less effective for RACS over the private L2 scheme. In other words, in the RACS scheme, cache blocks are turned on for the time more than in the private scheme when using the cache decay technique, decreasing the leakage energy savings.

The proposed LACS scheme simultaneously considers the performance and the leakage power consumption when determining a peer L2 cache to store an evicted block. The leakage energy savings is achieved by avoiding writing evicted blocks to peer L2 cache blocks that can be turned off. Combining this approach with the existing RACS scheme, we can achieve the leakage energy savings as well as the performance improvement.

3.2 Leakage-Aware Selection

In LACS, when evicting a block from a local L2 cache, it is decided where to write the evicted block by simultaneously considering the performance and the leakage energy consumption. The performance is considered in a similar fashion as in the existing RACS scheme. In order to con-

sider the leakage energy consumption, all the peer L2 caches decide whether receiving the evicted block will increase their leakage energy consumption or not. After the evaluation, each peer L2 cache sends its decision to the requesting L2 cache. In order not to increase the leakage energy consumption when a cache sharing technique is employed, the length of a dead time interval of a block should be kept as long as possible. This can be achieved by not waking up a turned off cache block or not preventing a cache block from switching to a sleep state. Therefore, in our proposed scheme, peer L2 caches check if they have following two kinds of blocks at the bottom of LRU stacks: the turned off L2 cache blocks which are likely to exit from the sleep state soon, referred as B_{off} blocks, and the turned on L2 cache blocks which will not enter a sleep state in a short time, referred as B_{on} blocks.

3.2.1 B_{off} Block Selection

When the bottom block of the L2 LRU stack is a turned off block, we decide if it is a B_{off} or not. B_{off} blocks are considered as replacement candidates in order not to wake up cache blocks which can be turned off for a long time. We decide that the cache blocks should be woken up and store the evicted block when they meets one of the following two conditions:

Condition 1. $C_{remain\_dead} \leq 0$

Condition 2. $C_{remain\_dead} * E_{cache\_block\_leak} + E_{cache\_dyn} \leq L_{mem} * (E_{cache\_leak} + E_{mem\_leak}) + E_{mem\_dyn}$

where

C_{remain_dead}: the number of remaining dead cycles
E_{cache_dyn}: dynamic energy consumption per cache access
$E_{cache\_block\_leak}$: leakage energy consumption of a cache block per cycle
E_{cache_leak}: leakage energy consumption of a cache per cycle
E_{mem_leak}: leakage energy consumption of a memory per cycle
E_{mem_dyn}: dynamic energy consumption per memory access
L_{mem}: the off-chip memory latency

Condition 1 checks if a turned off block is likely to be woken up in a short time and Condition 2 checks if the benefit of waking up a turned off block is greater than or equal to the cost of it in terms of the energy consumption. In both conditions, we should predict the number of remaining cycles until a turned off block is likely to be woken up, C_{remain_dead}, which can be obtained by subtracting the time from the last hit occurs to the cache block from a predicted dead time, $D_{prediction}$. We predict $D_{prediction}$ based on the history of dead time intervals because it is observed that most of the dead time intervals is in a certain range as can be seen in Figure 2. For instance, for CHOLESKY, 80% of the cache blocks have dead time intervals shorter than 10000K cycles and for FMM and OCEAN, more than 90% of the dead time intervals are within 10000K cycles. Therefore, we can predict $D_{prediction}$ as follows:

$$D_{prediction} = \frac{D_{prediction} + D_{interval}}{2},$$

where $D_{interval}$ is the last dead time interval of the replaced block. This might not predict the dead time interval exactly because it is predicted based on the history of entire private L2 cache blocks. However, it can predict the dead time interval of a L2 cache roughly because the dead time interval depends on the characteristics of a program. Furthermore, even when it has a wrong prediction value, it does not cause a critical performance loss because it only prevents an evicted block from being written to a peer cache. It is shown that employing this prediction technique works well in experimental results.

In Condition 2, if the benefit of waking up a peer L2 cache block is greater than or equal to the cost of it, we decide to turn the block on and store an evicted block. The left and right side of the Condition 2 equal the cost and the benefit when an evicted block is written to the turned off block of a peer L2 cache, respectively. Since writing the evicted block causes the peer L2 cache block to be turned on, the cost consists of the leakage energy consumption during C_{remain_dead} and the dynamic energy consumption per cache access, E_{cache_dyn}. However, E_{cache_dyn} can be ignored because it is very small compared to the leakage energy consumption. On the other hand, if an evicted block is written to a peer L2 cache and reused, the energy consumption could be decreased by avoiding an access to the off-chip memory because it can decrease the leakage energy consumption of both cache and memory during the access latency of the off-chip memory and the dynamic energy consumption of a off-chip access. The right side of the Condition 2 can be constant if we assume that L_{mem} and E_{cache_leak} always have the same values even though L_{mem} varies depending on congestion of a shared bus and E_{cache_leak} varies depending on the number of turned off blocks. It is reasonable because C_{remain_dead} on the left side of the equation is usually much larger than L_{mem}.

For the LACS scheme to decide a peer L2 cache to store an evicted block, each cache has $D_{prediction}$ value and it is updated every time the turned off block is replaced and woken up. To obtain the last dead time interval of the replaced block, $D_{interval}$, we modify the cache decay interval counters in [7]. After a cache block has not been accessed for a time-out threshold cycles, the cache block enters a sleep state but the corresponding decay counter keeps incrementing until a new block is brought. However, to distinguish the turned off cache block, we add a 1-bit per each cache block, on_off bit. Therefore, when the turned off block is replaced with the new block, the value of the decay counter indicates $D_{interval}$ and $D_{prediction}$ is updated using the new $D_{interval}$. Only when the decay counter of a turned on block gets saturated to a time-out threshold, it enters a sleep state. For turned off cache blocks, even when the decay counter reaches the time-out threshold, the state of cache blocks is not changed.

3.2.2 B_{on} Block Selection

When the bottom block of the L2 LRU stack is a turned on block, we decide if it is a B_{on} or not. B_{on} blocks are decided as replacement candidates because they are not likely to switching to a sleep state soon. The decision is made based

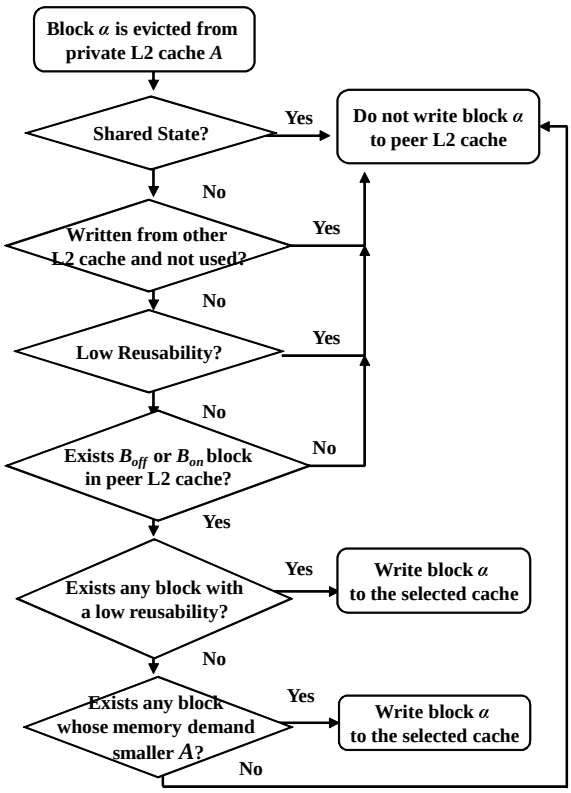

Figure 4: Process of leakage-aware cache sharing technique.

on the condition as follows:

$$C_{from\_last\_hit} < \tau/2$$

$C_{from\_last\_hit}$ is elapsed cycles from the last hit occurs and τ is a time-out threshold value to enter a sleep state. We decide that if $C_{from\_last\_hit}$ is less than half of the τ, the corresponding block is not likely to be turned off in a short time. If the above condition meets, our approach prefers to replace the peer L2 cache block with the evicted block.

3.3 Process of Leakage-Aware Cache Sharing Technique

Figure 4 summarizes the LACS steps in processing an evicted L2 cache block α. When a block α is evicted from a private L2 cache, LACS decides that it should not be written to any peer L2 cache in following cases: if the state of that block is shared, because it means that the same block is present in another L2 cache; if the block was transferred from a peer L2 cache but was not reused while residing in the peer cache, because such blocks have already had a chance to be reused; if the reusability of the evicted block is low, which is determined by corresponding ATIF pattern counter. If the block is not in above cases, we check if there exist peer L2 caches replying that they have B_{off} or B_{on} blocks. If peer L2 caches have such blocks, we look for a peer L2 cache block with low reusability. When there are more than one peer cache block with the low reusability, LACS randomly chooses one of them. If, however, peer L2 caches do not have the block with low reusability, LACS decides whether the evicted block will remain on-chip from the memory demand prediction. Otherwise our approach evicts the blocks out of the on-chip space.

The process of deciding where to write evicted blocks requires peer-to-peer communication lines. If an evicted block has a high reusability, its cache sends the set number of the evicted block, $Repl_{history}$ to all the peer L2 caches. Then the peer L2 caches send three bits of information on reply: one bit indicates whether the cache has a block with low reusability at the bottom of its LRU stack; and another bit indicates whether the value of $Repl_{history}$ for that cache is larger than the broadcast value of $Repl_{history}$. The other bit indicates that they have either a B_{off} block or a B_{on} block. Using these information, our proposed approach decides a peer L2 cache where an evicted block is written according to the process explained above. Writing to a peer L2 cache does not cause a subsequent write to the other peer L2 cache to avoid a ripple effect.

3.4 LACS Technique for Coherent Caches

Our proposed scheme is based on a private L2 cache organization which uses inclusion property between lower levels, such as L2 or L3, and higher level caches, such as L1. Inclusion property, proposed in [1], is usually used in multi-level caches to implement a cache coherence efficiently. In an inclusive cache, the cache block which is present in higher level caches should be present in lower level caches. Then, by keeping the states of the blocks of the higher level cache in the lower level cache, only the tags and the states kept in the lower level cache are checked when cache coherence protocol messages are received. Otherwise, both of the higher and the lower level caches might receive a large number of queries from a snooping bus, which causes a significant performance degradation to the higher lever cache.

We employ a MESI protocol to maintain cache coherence between processors. To apply the cache decay technique in an inclusive private L2 cache, we keep the tags and the states of a turned off block active in L2 caches for correctness. This could keep L1 caches from snooping traffic as in a private L2 cache organization which does not use the cache decay technique. Even though the LACS scheme does not turn off the tags and the states, it could reduce leakage energy significantly because the energy consumption of the tags and the states is relatively smaller than that of data blocks.

3.5 Hardware Overhead

The previous RACS scheme has hardware overhead compared to a pure private L2 cache organization because it requires additional counters for the three prediction schemes, reusability, memory demand, and leakage energy saving, and peer-to-peer communication lines between the L2 caches. Predicting reusability involves a 4-bit counter and a 2-bit counter at each block, to record the number of accesses with long and short time intervals, respectively. Additional 2 bits are required for each set to distinguish between long and short time interval accesses. These bits record the most recently accessed block of each set. In addition, each block needs 2 bits to indicate which processor writes it and a further bit indicates whether the block has been reused or not. For each private L2 cache, we also need 16 10-bit pattern counters. To predict the memory demand, a 8-bit counter

Table 1: Processor/Cache/Memory Configuration

Processor	4 Processors in-order
L1 D-Cache	32KB, 1-way 32B block, 1 cycle latency
L1 I-Cache	32KB, 1-way 32B block, 1 cycle latency
L2 Private Cache	512KB, 4-way 128B block, 8 cycle latency
Shared Bus	4bytes bus width, pipelined
Off-Chip Memory	300 cycle access latency

is used to record the time from the last replacement and another 8-bit counter records the replacement history. This comes to a total of 9 bits per block, 2 bits per set, and about 22 bytes per cache.

We use also several additional registers for LACS. Each cache has the three registers which store 16-bit $D_{prediction}$, 4-bit $E_{cache\_block\_leak}$, and 10-bit constant value for the benefit of writing the evicted block. We take a logarithm of the $E_{cache\_block\_leak}$ with base 2 to use a shifter instead of a multiplier. We add a 1-bit per each block, which distinguishes whether the block is turned on or not. The overall hardware overhead of the LACS scheme is about 1% of the area of a private L2 cache, which is effectively negligible. We note that the decision whether to write an evicted blocks to a peer L2 cache is not on a critical path because it can be made after the block is evicted from its original cache and placed in a write queue.

In addition, there is an area overhead to implement the cache decay mechanism that our approach is based on. We use two levels of counters to keep track of the cycles elapsed since each block was last accessed because decay intervals are usually tens of thousands of cycles as in [7]. It uses a single global cycle counter that is used to provide a tick for smaller local counter which exists per cache block. In our experiments, the global counter sends a tick to the local counters every 1000 cycles and 16-bit counters are used for the local counters per each block. We observed empirically that a 16-bit local counter is enough to record $C_{from\_last\_hit}$ and $D_{interval}$. In the experiment, we use 4 million cycles for τ, which is used for the L2 cache in [7]. This causes the 3% area increase [11] of the cache cells due to the logic required to turn off a cache block.

4. EXPERIMENTAL RESULTS

4.1 Simulation Environment

We modified the CATS [9] multiprocessor simulator to evaluate our technique. Cache-to-cache transfer of the cache block among private L2 caches is applied in all schemes and a MESI protocol is used for cache coherency. Table 1 shows the processor, cache, and memory configuration we used for experiments while Table 2 summarize the key power parameters for L2 cache and main memory. These parameters were estimated from CACTI 4.1 [5] using 70nm CMOS technology and SDRAM power estimation tool provided by Micron [10].

We evaluated 5 schemes as shown in Table 3. $Private_{no\_decay}$ and $RACS_{no\_decay}$ are the schemes which do not use the cache decay technique. while $Private_{decay}$ and $RACS_{decay}$ are the schemes which use the cache decay technique. We evaluated our scheme with 5 benchmarks, CHOLESKY, FMM, FFT, RADIX and OCEAN in SPLASH2 [13].

Table 2: L2 Cache and Memory Energy Parameters

Private L2 Cache	Dynamic read	0.11 nJ
	Dynamic write	0.01 nJ
	Leakage power	1802 mW
Off-Chip Memory	Dynamic read/write	2 nJ
	Standby Power	50 mW

Table 3: Evaluated Schemes

$Private_{no\_decay}$	a cache decay policy is not used a cache sharing technique is not used
$RACS_{no\_decay}$	a cache decay policy is not used a cache sharing technique is used considering reusability
$Private_{decay}$	a cache decay policy is used a cache sharing technique is not used
$RACS_{decay}$	a cache decay policy is used a cache sharing technique is used considering reusability
$LACS$	a cache decay policy is used a cache sharing technique is used considering reusability & leakage energy

4.2 Experimental Results

Figure 5 shows the normalized energy consumption. The energy consumption of each scheme is normalized to the $Private_{no\_decay}$ scheme. We considered both the dynamic and the leakage energy consumption of the L2 cache and the memory. The dynamic energy consumption includes the energy due to cache accesses, the extra L1 misses caused by turning off the block too early, and the overhead due to additional structures for LACS. The dynamic energy consumption due to the additional counters is obtained by multiplying the number of bits of the counters by the dynamic energy consumption of a bit of the cache which is also calculated by CACTI 4.1. The leakage energy consumption is also considered. It consists of the leakage energy of cache and memory, the extra leakage overhead due to the $Gated-V_{dd}$ technique

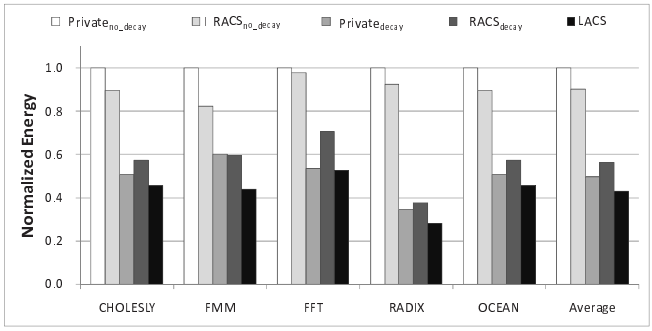

Figure 5: Normalized energy consumption.

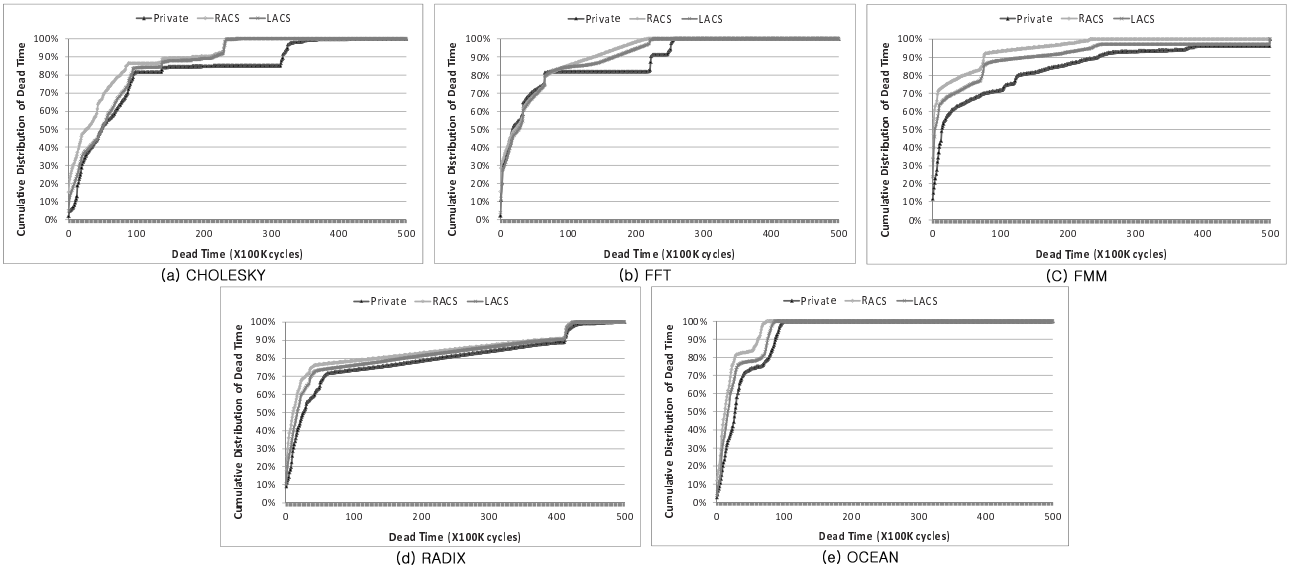

Figure 6: Cumulative distributions of dead times.

[11] used to turn off a cache block, and the overhead due to additional counters. We calculated the energy of the additional counters by assuming that each bit of them, described in Section 3.5, has the same leakage consumption as a bit in the L2 cache.

As can be seen in Figure 5, the energy consumption of $RACS_{decay}$ is bigger than $Private_{decay}$, even though the performance of $RACS_{decay}$ is higher than that of $Private_{decay}$. The reason why the $RACS_{decay}$ has a larger energy consumption is because dead time intervals of cache blocks get shorter as explained Section 3.1. However, LACS reduces the energy consumption by 13.4% and 23.6% on average over the $Private_{decay}$ and $RACS_{decay}$, respectively, while keeping the performance almost same as $RACS_{no\_decay}$. It is achieved by writing evicted blocks to peer L2 caches in the way that the dead time intervals of cache blocks are kept similar to the $RACS_{decay}$ scheme, thus reducing the leakage energy consumption. Figure 6 shows this analysis using the cumulative distributions of dead times of each scheme. In most cases, the cumulative distribution of the LACS scheme is placed between the cumulative distribution of the $RACS_{decay}$ scheme and that of the $RACS_{decay}$ scheme. This means that the fraction of short dead time intervals decreases and that of long time intervals increases over the $RACS_{decay}$ and also shows that our proposed dead time prediction technique works well. Even though writing evicted blocks selectively with consideration of the leakage energy decreases the total number of blocks written to peer caches and degrades the performance improvement compared over the $RACS_{no\_decay}$ scheme, it can reduce the energy consumption significantly with a small performance loss.

Figure 7 shows the execution time of the benchmarks normalized to the $Private_{no\_decay}$ scheme. $RACS_{no\_decay}$ reduces the execution time by 9.6% on average compared to $Private_{no\_decay}$. Especially in CHOELSY and FMM, the performance improvement is up to 8% and 13.6%, respectively, because many blocks are reused after the eviction. In $Private_{decay}$ and $RACS_{decay}$, the performance degradation due to the extra misses is less than 1% over $Private_{no\_decay}$ and $RACS_{no\_decay}$ except for RADIX and OCEAN. For RADIX, the performance degradation of $Private_{decay}$ over $Private_{no\_decay}$ is about 3% while the performance of $RACS_{decay}$ is almost same as $RACS_{no\_decay}$. This is because many of the blocks cannot be turned off in $RACS_{decay}$ compared to the $Private_{decay}$. Writing evicted blocks to peer L2 caches prevents cache blocks from entering a sleep state. In LACS, the performance improvement is a little smaller than the $RACS_{no\_decay}$ scheme, especially for FMM. This is because the number of blocks which are written to peer caches is reduced when the LACS scheme is applied. However, the performance loss over the $RACS_{no\_decay}$ scheme is small because the LACS scheme can write an evicted block to one of the peer caches in most cases.

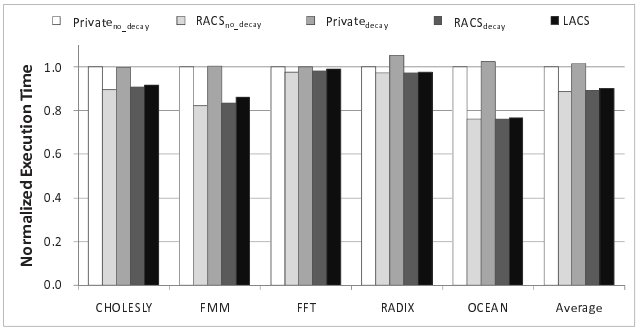

Figure 7: Normalized execution times.

Figure 8 shows the normalized energy delay product of each scheme. Our proposed scheme reduces the energy delay product by 26.4% and 18.6% on average over $Private_{decay}$ and $RACS_{decay}$, respectively. Consequently, LACS shows the best result because it can reduce the energy consump-

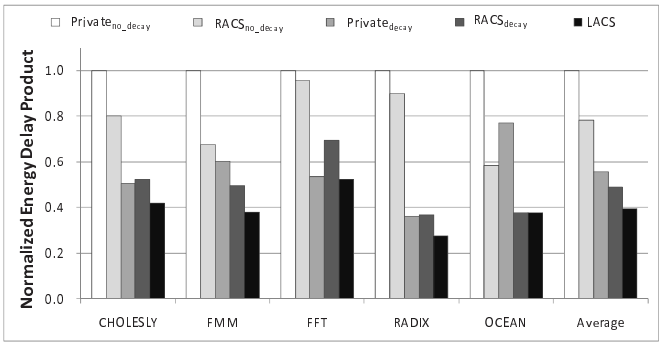

Figure 8: Normalized energy delay product.

tion but the performance loss of it is relatively small.

5. CONCLUSIONS

We proposed a Leakage-Aware Cache Sharing technique called LACS which is based on a private L2 cache organization for CMPs. When a cache sharing technique is applied to a private L2 cache organization without leakage energy consideration, the dead time intervals of cache blocks decrease because writing evicted blocks to peer L2 caches might wake up turned off blocks or prevent cache blocks from entering a sleep state. Therefore, our proposed scheme checks if there exist following two kind of blocks among the blocks at the bottom of LRU stacks of peer L2 caches: the blocks which are likely to exit from the sleep state and the blocks which will not enter a sleep state in a short time. Among these blocks, then, the LACS scheme selects a peer L2 cache block with low reusability in the same way the previous RACS scheme proposed. By writing the evicted blocks only when such conditions are met, LACS reduced the energy consumption as well as improved the performance.

Experimental results show that the LACS scheme is efficient, improving the performance by 23.3% compared to the $Private_{no\_decay}$ scheme while reducing the energy consumption by 13.4% and 23.6% on average over $Private_{decay}$ and $RACS_{decay}$, respectively. Our LACS scheme also reduced the energy delay product by 26.4% and 18.6% on average over the $Private_{decay}$ and $RACS_{decay}$, respectively.

6. ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the National Research Lab. This work was also supported in part by the Brain Korea 21 Project in 2008. Program funded by the Ministry of Education, Science and Technology (No.R0A-2007-000-20116-0). The ICT at Seoul National University provides research facilities for this study.

7. REFERENCES

[1] J. Baer and W.H.Wang. On the inclusion properties for multi-level cache hierarchies. In *International Symposium on Computer Architecture*, June 1998.

[2] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *International Symposium on Microarchitecture*, December 2004.

[3] J. Chang and G. S. Soh. Cooperative caching for chip multiprocessors. In *International Symposium on Computer Architecture*, July 2006.

[4] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in cmps. In *International Symposium on Computer Architecture*, June 2005.

[5] T. David, T. Shyamkumar, and J. Norman. Cacti 4.1. In *http://www.hpl.hp.com/personal/Norman Jouppi/cacti4.html*. HP, 2006.

[6] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *International Symposium on Computer Architecture*, May 2002.

[7] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *International Symposium on Computer Architecture*, May 2001.

[8] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Architectural Support for Programming Languages and Operating Systems*, 2002.

[9] D. Kim, S. Ha, and R. Gupta. Cats: cycle accurate -driven simulation with multiple processor simulators. In *Design, Automation and Test in Europe*, March 2007.

[10] Micron. Calculating memory system power for ddr. In *Technical report*, 2005.

[11] 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 *International Symposium on Low Power Electronics and Design*, May 2000.

[12] E. Speight, H. Shafi, L. Zhang, and R. Rajamony. Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors. In *International Symposium on Computer Architecture*, June 2005.

[13] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerati. In *International Symposium on Computer Architecture*, June 1995.

[14] S. Youn, H. Kim, and J. Kim. A reusability-aware cache memory sharing technique for high-performance low-power cmps with private l2 caches. In *International Symposium on Low Power Electronics and Design*, August 2007.

[15] M. Zhang and K. Asanovic. Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *International Symposium on Computer Architecture*, June 2005.