Performance Optimization for Low-Leakage Caches based on Sleep-Line Access Density

Komiya, Reiko
Department of Electronics Engineering and Computer Science, Fukuoka University

Inoue, Koji
PRESTO, Japan Science and Technology Agency

Murakami, Kazuaki
Institute of Systems & Information Technologies/KYUSHU | Department of Informatics, Kyushu University

出版情報：Workshop on Optimizations for DSP and Embedded Systems. 4, 2006-03-26
バージョン：
権利関係：

Performance Optimization for Low-Leakage Caches based on Sleep-Line Access Density

Reiko Komiya[†1, †2] Koji Inoue[†3, †4] Kazuaki Murakami[†2, †3]

†1 Department of Electronics Engineering and Computer Science, Fukuoka University
†2 Institute of Systems & Information Technologies/KYUSHU
†3 Department of Informatics, Kyushu University
†4 PRESTO, Japan Science and Technology Agency

arch-ccc-lpc”at”c.csce.kyushu-u.ac.jp

ABSTRACT
As the transistor feature sizes and threshold voltages reduce, leakage energy consumption has become an inevitable issue for high-performance microprocessor designs. In order to solve the energy issue, a number of techniques to reduce the cache leakage energy have so far been proposed. However, the low-leakage caches affect negatively the processor performance due to the accesses to non-state-preserving sleep-mode lines. In this paper, we analyze the access behavior on a low-leakage cache and show a remarkable observation for the density of sleep-line accesses. Based on this observation, we propose a new cache management technique to alleviate the performance degradation caused by low-leakage caches. In our approach, a small number of cache lines which are frequently accessed in the sleep mode are forced to stay in always-active mode. Although this mode is high leakage, it saves the state. Thus, the performance overhead caused by the leakage optimization can be eliminated.

Keywords
High performance, Low leakage, Cache

1. INTRODUCTION
Due to the popularization of battery operated devices such as laptop and hand-held computers, energy consumption has become a key constraint in microprocessor designs. The energy dissipated in CMOS circuits consists of two parts: dynamic energy and static (or leakage) energy. The dynamic energy is consumed by charging and discharging load capacitances in circuits, while the leakage energy is wasted by leakage current in non-ideal transistor operations, i.e., incomplete turning off. With the increasing number of transistors employed in a chip and the continuing reduction in threshold voltages of these transistors, leakage energy has become a major concern. Especially, reducing the leakage in on-chip caches is essential, because of a tendency that a significant portion of transistor budget in microprocessors is invested to on-chip memories. For example, in the case of a 0.07μm process technology, it has been estimated that leakage energy accounts for 70% of total cache energy [5].

To challenge the leakage issue, a number of researchers have proposed efficient leakage reduction techniques ([2]~[4], [5]~[12], [15] [16]). To reduce the cache leakage, it is required to support at least two operation modes; a conventional high-leakage mode (or **active mode**) and an optimized low-leakage mode (or **sleep mode**). Furthermore, there are two options to implement the sleep mode; non-state-preserving and state-preserving. The former gates the supply voltage to SRAM cells, thus large amount of leakage is reduced, but performance is negatively affected due to losing the data in the SRAM cells [5][11]. While the latter can be implemented by optimizing the supply voltage as DVS ([2][7]) or the transistor threshold voltage as VT-CMOS [3]. This approach can maintain the cache-hit rates of non-optimized conventional caches. In state-preserving, the sleeping line needs to be transited to the active mode before the SRAM access is started. It can alleviate the performance impact, but leakage reduction is not as high as the non-state-preserving scheme. With the advances of VLSI technology, changing the supply voltage at run time may become harder and harder due to the effects of wire delay, Vdd noise, and so on. Furthermore, the speed of microprocessors tends to be increased. As the results, it becomes difficult to materialize low supply voltage which does not destroy for state-preserving. In addition, the reference [9] reports that the state-preserving scheme is weak to the soft error. Therefore, we expect that the non-state-preserving approach becomes mainstream, and focus on this approach in this paper.

The non-state-preserving approach stops the supply voltage of corresponding line and ruins the data. Thus, a miss is surely made in the reference to a sleep mode line. If the data referred to again is lost, the number of miss increases as compared with the conventional cache which does not use the leakage reduction technique. It causes the

performance degradation of the processor. We call this miss "**Sleep-miss**". In in-order execution, sleep-miss inflicts heavy damage on the processor performance, because the next instruction cannot be executed. In our evaluation, we have noticed that it has been observed that in the worst case, the processor performance is degraded by 37%.

To solve the performance problem in low-leakage caches, in this paper, we propose a new cache management technique. First, we analyze the characteristics of access behavior on a low-leakage L1 data cache and show that some cache lines have extremely high degree of sleep-miss density, i.e., a small number of lines are responsible for the majority of sleep-miss. Based on this observation, we propose a new operation mode, called always-active mode. In our approach, the cache lines which cause frequently the sleep-misses are forced to stay in the active mode. We then evaluate the energy-performance efficiency of the always-active scheme. As a result, it is observed that in some benchmarks our technique can hide almost all the performance degradation caused by a conventional Cache Decay [5] without affecting the energy efficiency.

The rest of this paper is organized as follows: In Section 2, we explain the experimental environment. Section 3 analyzes the detail of sleep-miss behavior. Section 4 proposes the always-active optimization and discusses its implementation alternatives. We then evaluate the performance-energy efficiency of the proposed method in Section 5. Section 6 shows related work. Finally, in Section 7, we conclude this paper.

2. Experimental Setup

A cache simulator is developed to estimate the cache performance and energy. The processor configuration assumes an embedded processor, because the performance degradation is large when the leakage reduction technique is applied. More specifically, the embedded processor "Xscale" is assumed as shown in Table 1. Data L1 cache is scope of application of the leakage reduction technique. We use the SimpleScalar simulator tool set [13], and twelve of integer and fourteen of floating-point programs from the SPEC CPU 2000 benchmark set [14]. In the cache simulation, the first one billion instructions are skipped to make the execution stable, and the following 500 million instructions are used for the measurement.

"Cache decay" is applied as the traditional leakage reduction technique. Cache decay is a famous technique to reduce leakage energy [5]. Although our technique can be applied to other non-state-preserving low-leakage caches, we use the Cache decay as the base model. In the Cache decay strategy, accesses to each cache line are monitored. If there are no accesses to a line during a given period, called **decay interval**, the cache predicts that the contents

Table 1: Configuration of Processor

Instruction issue	In-Order
Instruction decode width	2 instructions / clock cycle
Instruction issue width	2 instructions / clock cycle
Cache memory	
L1 data	32KB (32B/entry, 32way, 1K entries)
L1 instruction	32KB (32B/entry, 32way, 1K entries)
Hit latency	
L1 cache	1 clock cycle
Main memory	32 clock cycles
Memory bandwidth	8B
Memory ports	1
ITLB, DTLB	
Entry	1M entries (4KB/entry, 32way, 32 entries/way)
Miss penalty	30 clock cycles

of the line are not expected to be reused. The supply voltage to the decayed lines is gated. In point of fact, decay interval is counted using hierarchical counter mechanism [5]. A global counter counts up to 1/4 decay interval with each cycle. Each line has local two-bit counter. When global counter is saturated, all local counters are incremented. If a two-bit counter reaches its maximum, a cache line transits to the sleep mode. A local counter is reset when the line is accessed. For the rest of this paper, we call a cache line in the sleep mode a **sleep-line**. Similarly, a line in the active mode is referred to as an **active-line**. In addition, decay interval is assumed 4K clock cycles.

3. Analyzing Sleep-Line Access Behavior

In order to achieve cache leakage reduction without hurting microprocessor performance, it is very important to employ an efficient mode transition control. If the cache attempts to transit aggressively into the sleep mode, large amount of leakage is reduced in turn. However, this scenario may affect negatively the microprocessor performance due to the occurrence of frequent sleep-miss. Therefore, we analyze Sleep-line accesses which increase the cache misses in this section.

3.1 Performance Impact of Sleep-Line Access

We have evaluated the impact of the sleep-miss on microprocessor performance. Figure 1 shows the program-execution time in terms of clock cycles for Cache decay. These values are normalized by the execution time of conventional cache without low leakage technique. Increasing in the execution time for f179.art and i176.gcc is

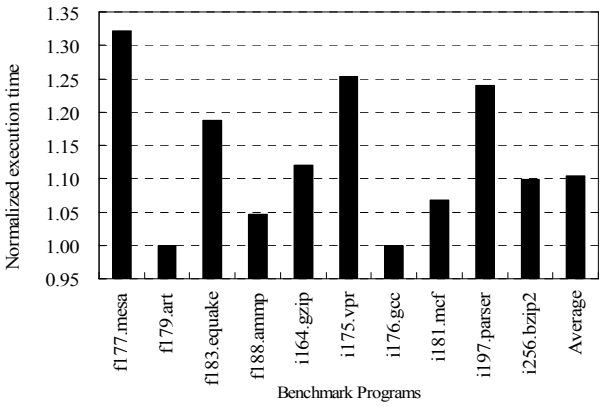

Figure 1: Normalized Execution Time of Cache decay

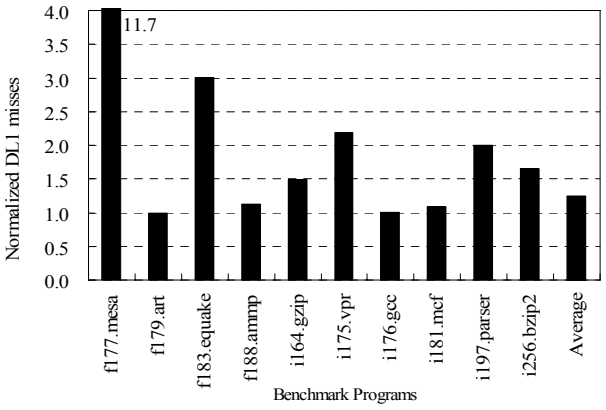

Figure 2: Normalized DL1 Misses

trivial. However, another benchmark programs make execution time longer. Performance overhead becomes 32.1% in worst case. Thus, we believe that challenging the performance issue of low-leakage caches is worthwhile.

Next, we represent the number of Data L1 cache misses in Figure 2. The y-axis shows the normalized DL1 misses by non-optimized conventional cache. If the value becomes one or more, it means sleep-miss which Cache decay caused. This figure tells us that performance degradation goes up in proportion to the number of sleep-miss. If we can cut back these misses, we will obtain an advantage of the performance improvement.

3.2 Sleep-Miss Density

In general, the memory references have spatial locality. Therefore, it is expected that there is spatial locality also in sleep-miss accesses. We refer to the frequency of sleep-miss accesses to each cache line as **sleep-miss density** (***SMD***). The SMD of line *i* is defined as follows:

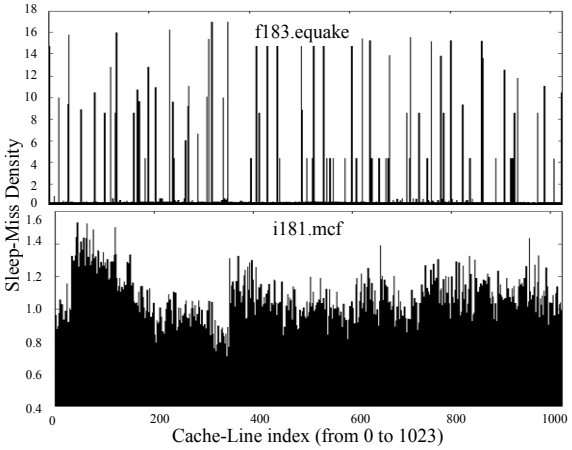

Figure 3: Sleep-miss Density (f183.equake, i181.mcf)

$$SMD_i = \frac{N_{sleep-miss(line-i)}}{N_{sleep-miss(avg)}} \quad (1)$$

where $N_{sleep\text{-}miss(line\text{-}i)}$ is the total number of sleep-miss accesses occurred at the cache line *i*, and $N_{sleep\text{-}miss(avg)}$ is the average number of sleep-miss accesses for all cache lines. Namely, if the *SMD* value of a cache line is 2.0, it means that the line causes the double of sleep-miss accesses compared with the average.

Based on the setup stated with section 2, we measured SMD_i in each line. Figure 3 shows the *SMD* of each cache line for two benchmark programs. The x-axis shows the cache-line index in the assumed 32KB 32-way cache. For i181.mcf, many cache lines have the *SMD* value of around 1.0. Actually, the *SMD* value of all line is smaller than 1.6. On the other hand, for f183.equake, we see that some cache lines indicate much higher degree of *SMD*. Figure 4 presents the breakdown of cache lines in terms of the *SMD*. The five programs, f179.art, f188.ammp, i175.vpr, i197.parser and i256.bzip2 show the same characteristics with f183.equake. Namely, the *SMD* value of almost all the cache lines is less than 1.0, while that of a few lines (less than 10%) is equal to or greater than 4.0. Figure 5 reports the breakdown of sleep-miss accesses, that is, how much the sleep-miss accesses are dominated by the cache lines having different values of the *SMD*. For all benchmark programs, the cache lines indicating higher degree of *SMD* (equal or greater than 1.0) dominate the total sleep-miss accesses.

From the observations explained above, we can consider that in many cases a small number of cache lines are responsible for the majority of sleep-miss accesses. For instance, in f183.equale, the cache lines with $SMD \geq 4.0$ are only 7.7%, but they account for 75.2% of sleep-miss accesses. On average for all benchmarks, 2.6% of cache lines have $SMD \geq 4.0$, and they cause 25.2% of total sleep-

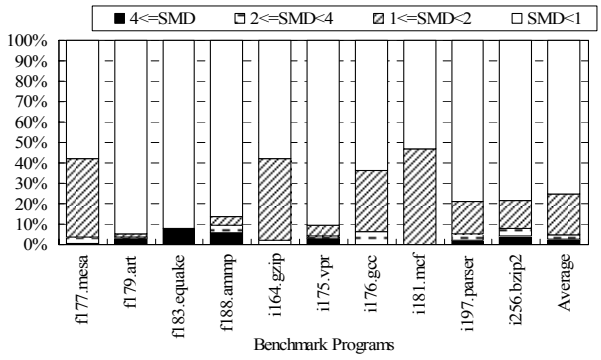

Figure 4: Breakdown of Lines in terms of *SMD*

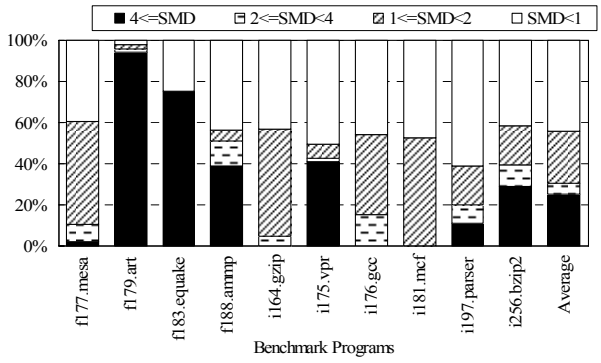

Figure 5: Breakdown of Sleep-Miss Accesses

miss accesses. Furthermore, if we focus on the range of *SMD*>1.0, 55.7% of sleep-miss accesses are caused by 24.5% of cache lines.

4. Supporting Always-Active Mode

4.1 Run-Time Operation-Mode Optimization

Each cache line operates as depicted in Figure 6. In the case of Cache decay, the line transits to the sleep mode at the end of every decay interval, and enters into the active mode when it is accessed. Correspondingly, the state of an "**always-active mode**" is added in the method which we propose. This mode prohibits a cache line to transit to the sleep mode. More specifically, even if no-access time goes through decay-interval, it does not change to sleep-mode. A cache line operating in the always-active mode is referred to as an **always-active line**. By assigning the cache lines which causes frequently the sleep-miss accesses to the always-active mode, we can alleviate the negative impact of performance. Not only that, we can also reduce the reference energy to a low level memory.

How many cache lines should be assigned to the always-active mode are most important considerations. Although increasing the number of always-active lines improves the performance, the energy efficiency may be degraded. On

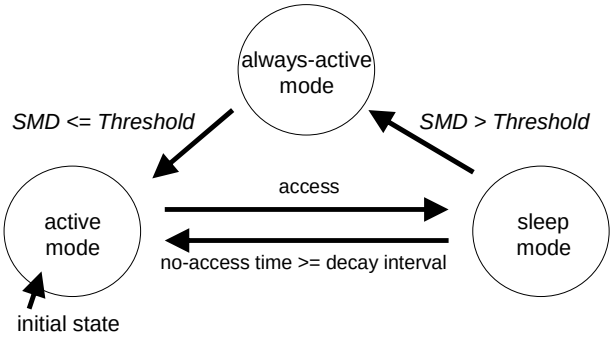

Figure 6: State Diagram for Operation Mode

the other hand, the lack of the always-active lines may not be able to compensate for the negative performance impact caused by the sleep-miss. For this design alternative, we exploit the value of *SMD*. If a cache line indicates the value of *SMD* exceeding a given threshold, the line is considered to be responsible for the majority of sleep-miss accesses. Therefore, that line is assigned to the always-active mode. Otherwise, the line is managed as active mode. Concretely speaking, only if one of the following equations is approved, the operation mode of that line is transited to always-active mode.

$$SMD_i > Threshold \quad (2)$$

where *Threshold* is a given parameter in order to resolve always-active line. We substitute the equation (1) to (2).

$$\frac{N_{sleep-miss(line-i)}}{N_{sleep-miss(avg)}} > Threshold \quad (3)$$

This equation shows that it is necessary to get $N_{sleep\text{-}miss(avg)}$ and $N_{sleep\text{-}miss(line\text{-}i)}$, in order to realize always-active mode.

4.2 Hardware Implementation

In this section, we consider the implementation for supporting the always-active mode. The cache which supports always active mode is presented in Figure 7. This cache assumes 1 way and 1024 lines. The domain surrounded with the broken line is cache decay. The cache decay has 1-bit decay flag which indicates operating state, 2-bit local counter for each cache line, and one global counter The local counter is incremented when the global counter exceeds 1/4 decay interval clock cycles. And, the decay flag is set to one if the local counter is saturated, i.e., the supply voltage provided to the associated line is gated. If sleep-line is referred, the corresponding decay flag is reset, in order to change it to active-line.

On the other hand, the SMD_i value of each line needs to be calculated in order to determine always-active line dynamically during program execution. Formula (3) is rewritten as follows:

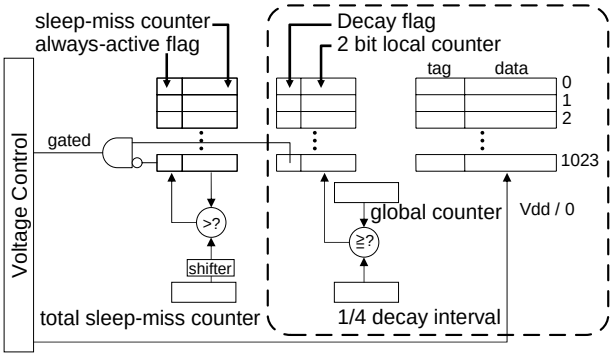

Figure 7: Supporting Always-Active Mode

$$N_{sleep-miss(line-i)} > Threshold \times N_{sleep-miss(avg)} \quad (4)$$

$$Threshold \times N_{sleep-miss(avg)} = N_{sleep-miss(total)} >> N_{shift-bit} \quad (5)$$

$$N_{shift-bit} = \log_2(N_{line}) - \log_2(Threshold) \quad (6)$$

where $N_{sleep\text{-}miss(total)}$ is the total number of sleep-miss, $N_{shift\text{-}bit}$ is the bit count which should be shifted, and N_{line} is the number of lines in the L1 data cache. N_{line} and *Threshold* are already determined at the time of a layout. Namely, if we assume N_{line} is 1024 ($=2^{10}$) and *Threshold* is 2 ($=2^1$), $N_{shift\text{-}bit}$ is 9. Therefore, only the counter for $N_{sleep\text{-}miss(line\text{-}i)}$, and $N_{sleep\text{-}miss(total)}$, and shifter are required in order to implement always-active mode. Other circuits are not required any modification from the conventional cache decay. Concretely speaking, 1-bit flag (called **always-active flag**), 20-bit counter (called **sleep-miss counter**), a shifter, and **total sleep-miss counter** is needed.

Although not only a data but a tag is transited to sleep mode in conventional Cache decay, only a data is transited to sleep mode in always-active method. If the line hit to the tag is sleep-line, it will be judged that sleep-miss occurred. When a sleep-miss occurs, corresponding sleep-miss counter and total sleep-miss counter count up. Moreover, expression (4) is judged. If the sleep-miss counter is larger, the line is changed to always-active line because it caused performance degradation. In contrast to this, the line transits to the decay mode if the number of sleep-miss counter is smaller.

If the always-active flag is set to one, the associated cache line ignores the current status of the decay flag and operates in the active mode. In other words, the always-active flag masks the decay flag.

4.3 Energy Model

We define the energy model which varies with the leakage reduction technique, as follows:

$$E_{total} = LE_{L1} + DE_{L1} + DE_{main\_memory} \quad (7)$$

where LE_{L1}, DE_{L1}, and DE_{main_memory} are respectively, the leakage energy, the dynamic energy consumed in the data L1 cache, and the dynamic energy consumed with reference for main memory.

LE_{L1} is given by the following equation:

$$LE_{L1} = \sum_{j=0}^{CC} (LE_{bit} \times N_{active(j)}) \quad (8)$$

where *CC* is the program execution time in terms of clock cycles, LE_{bit} is the average leakage energy in one SRAM cell consumed per clock cycle, and $N_{active(j)}$ is the total number of bits which is operating active mode at *j* clock cycle. As active-line increases, $N_{active(j)}$ increases, while *CC* decreases.

Next, we focus on DE_{L1}. This energy can be categorized into 3 parts. The first is the energy consumed in the non-optimized conventional cache. The second is consumption energy of addition circuit for cache decay. The third is the energy of additional circuit required for implementation of always-active mode. Therefore, DE_{L1} is expressed as follows:

$$DE_{L1} = DE_{org} + DE_{decay} + DE_{aa} \quad (9)$$

Without applying the low-leakage technique, DE_{org} is constant. DE_{decay} depends on the *CC*, because global counter is incremented with each clock cycle. DE_{aa} increases in proportion to the number of sleep-miss.

Finally, DE_{main_memory} is given by the average energy per access to main memory $DE_{mm/access}$ and the total number of main memory reference N_{mm_access}, as follows:

$$DE_{main\_memory} = DE_{mm/access} \times N_{mm\_access} \quad (10)$$

Always-active mode allows the reduction of N_{mm_access}.

To evaluate E_{total}, five parameters ($DE_{org/access}$, $DE_{aa/access}$, $DE_{mm/access}$, $DE_{decay/access}$, and LE_{bit}) are assumed as follows:

- $\boldsymbol{DE_{org/access}}$ is the average dynamic energy per access to the conventional cache. To measure this, we use simulator CACTI 3.0 which computes cache access time and energy [1]. As a result of measuring, $DE_{org/access}$ = **1.90 nJ**. At this time, the CMOS technology is assumed as 0.07μm.
- $\boldsymbol{DE_{aa/access}}$ is dynamic energy per access to sleep-miss counter. **4.20 pJ** is obtained by the approximation using $DE_{org/acces}$ and the ratio of active bit counts per reference.
- $\boldsymbol{DE_{mm/access}}$ is dynamic energy per access to main memory. This parameter is defined as "$\boldsymbol{k*DE_{org/access}}$". We have investigated the "*k*" with various number, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. Because it

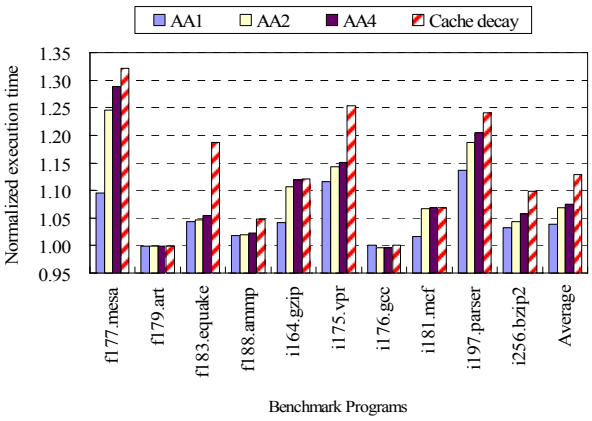

Figure 8: Normalized Execution Time of AA

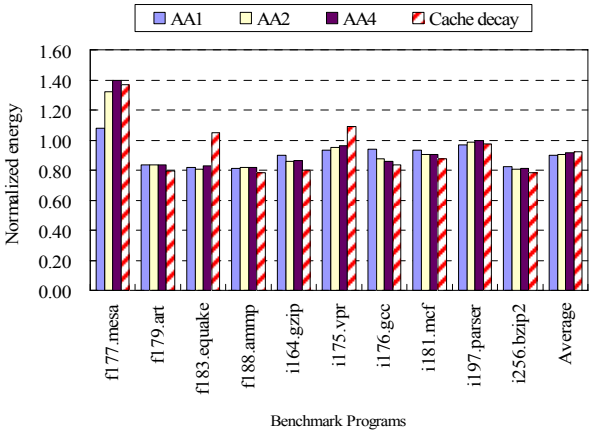

Figure 9: Normalized Energy of AA

cannot be discussed here for lack of space, it is assumed that $\boldsymbol{k}$ **= 20** in this paper.

- $\boldsymbol{DE_{decay/access}}$ is determined based on the design reports in [5]. The energy of 2 bit local counter is 0.1 pJ per access. In addition, it is assumed that decay interval is 4K clock cycle. Therefore, global counter needs 10 bits. In consequence, the energy of global counter is 0.5 pJ.
- $\boldsymbol{LE_{bit}}$ is decided in accordance with [2]. This paper reports the value of $DE_{org/access}$ and LE_{bit}. After considering these values and cache configuration, LE_{bit} is **0.13 pJ**.

5. Evaluation

In this section, we evaluate the energy-performance efficiency of the always-active low-leakage cache. The following caches are compared for the evaluation.

- **Cache decay**: a conventional cache decay
- **AA1/2/4**: a cache decay supporting the always-active mode. The number following to the string "AA" is the *SMD* threshold. This threshold means "*Threshold*" which is the variable of the expression (2). For instance, in AA4, the cache lines whose *SMD* value is equal to or greater than 4 are assigned to the always-active mode.

5.1 Performance Improvement

We have measured the program-execution time in terms of clock cycles for both the conventional cache decay and proposed always-active caches. Figure 8 shows the simulation results. Each execution time is normalized by the conventional cache which does not support any leakage reduction technique. For eight benchmarks, *f177.mesa, f183.equake, f188.ammp, i164.gzip, i175.vpr, i181.mcf, i197.parser,* and *i256.bzip2*, the proposed approach effectively compensates for the performance overhead caused by the cache decay. On the other hand, for *f179.art* and *i176.gcc*, the original decay approach can maintain the performance of non-optimized conventional cache. Even in such a case, the always-active scheme does not give any negative impact on the performance. On average, AA1, AA2, and AA4 reduce the performance overhead from 12.8% caused by the cache decay to 3.8%, 6.7%, and 8.4%, respectively.

5.2 Energy Reduction

We have measured the cache energy based on the energy model explained in Section 4.3. Figure 9 shows the normalized energy by non-optimized conventional cache. All the benchmarks used, except f177.mesa, achieve almost the same energy reduction rates as the cache decay. In the best case, f183.equake, our approach cut the energy 20% rather than cache decay. In f177.mesa, always-active technique also reduces the energy rather than cache decay. However, the energy overhead to non-optimized conventional cache is expensive.

5.3 Discussions

Based on the simulation results reported in Section 5.1 and 5.2, we discuss the energy-performance efficiency of the always-active approach. We can categorize the evaluation results into four cases.

In the first case, the always-active approach achieves high-performance and low-energy consumption, i.e., the cache reduces both energy and execution time rather than cache decay approach. For f183.equake, and i175.vpr belong to this category. Furthermore, although we see a small impact on the energy efficiency, large performance improvements can be achieved for some benchmarks, e.g., i197.parser. Figure 10, 11, and 12 depict the breakdown of dissipation energy of non-optimized cache, cache decay, and AA1, respectively. This category's benchmarks have the same characteristics. As compared with the

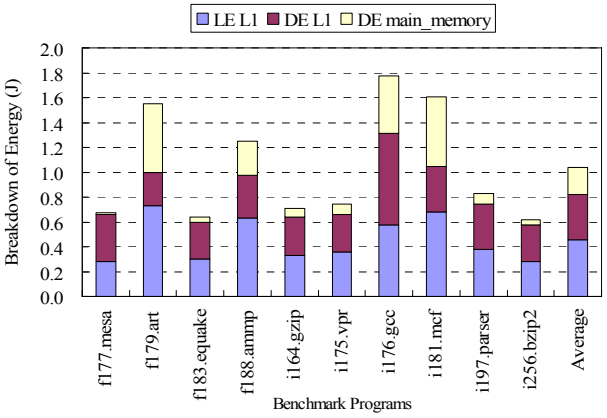

Figure 10: Breakdown of Energy for Non-Optimized Cache

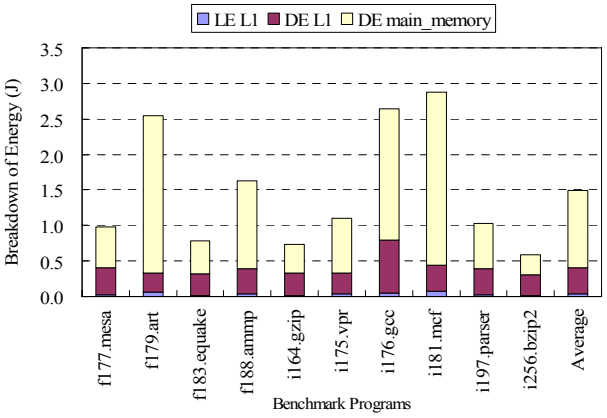

Figure 11: Breakdown of Energy for Cache Decay

conventional cache, DE_{main_memory} increases and LE_{L1} decreases with application of always-active technique. However, these benchmark programs have the very high reduction effect of DE_{main_memory} as compared with cache decay. In f183.equake, always-active technique increased LE_{L1} 84.8 mJ compared with cache decay, but, it reduced DE_{main_memory} 258 mJ. At this time, the performance has improved 14.5%. Additionally, only a small number of cache lines indicate higher values of the *SMD* and are responsible for the majority of sleep-miss accesses. For these applications, supporting the always-active mode is very effective.

The second case, the proposed cache aggressively reduces the sleep-miss penalty by sacrificing the energy efficiency. For f188.ammp, i164.gzip, i181.mcf, and i256.bzip2 show clearly this characteristic. This is because a number of cache lines in these benchmarks are allocated to the always-active mode. For i164.gzip, a number of cache lines have the *SMD* value less than 2 as shown figure 5. This situations increase the number of always-active lines,

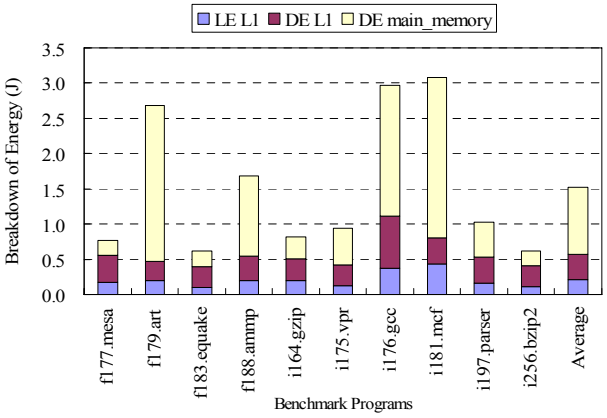

Figure 12: Breakdown of Energy for AA1

resulting in the similar effects with the non-optimized conventional cache. Since the influence of LEL1 increase is larger than the DE reduction effect, as compared with Cache decay, energy consumption of always-active is large.

In the third case, improves neither performance nor energy of the conventional cache decay. For instance, f179.art and i176.gcc fall under this category. As can be understood from figure 1, there are no performance degradation with cache decay, i.e. there is extremely little sleep-miss. These situations strongly restrict the number of the always-active lines, thus the proposed approach does not provide any advantages over the conventional cache decay.

From what has been said above, I have come to the conclusion that the profits of always-active are decided by the trade-off between the DE_{main_memory} reduction effect and the increase in LE_{L1}. Therefore, always-active technique has the ability to acquire the energy reduction and the high performance improvement by performing suitable line selection.

5.4 Effect of Decay Interval

The always-active method improves the execution time by reduction of sleep-misses. In order to reduce sleep-miss in cache decay, it is only necessary to lengthen the decay interval. Therefore, the execution time and energy at the time of setting decay interval to 4K, 8K, 64K, and 512K clock cycle are measured. Figure 13 shows the normalized execution time and normalized energy of AA1 and cache decay. Benchmark program is f183.equake. In the case of 512K decay interval, the energy reduction effect is not obtained, because the period until it changes to sleep-mode is too long. Correspondingly, if suitable decay interval is set, cache decay can reduce energy, without performance degradation. In f183.equake, 64K decay interval is the most appropriate. However, it is known that optimal decay interval differs for every program [10]. On the other hand,

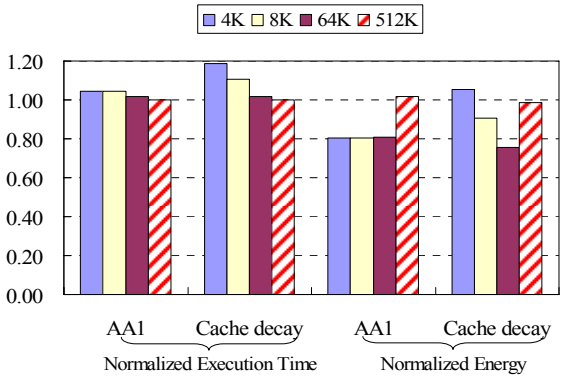

Figure 13: Effect of Decay Interval to Execution Time and Energy (f183.equake)

except for 512K decay interval, always-active method constantly reduces energy consumption and is also suppressing the increase of execution time.

6. Conclusions

During our investigations of cache decay behavior, there is a spatial locality of accesses to the turning-off lines. Based on this observation, we have proposed to support the always-active mode in order to achieve high-performance, low-leakage caches. In our approach, cache lines which caused the performance degradation are forced to stay in the high-speed but high-leakage mode.

We have evaluated the energy-performance efficiency of the proposed method. As a result, it has been observed that our approach can eliminate almost the performance overhead, while maintaining the energy reduction efficiency, compared with the conventional cache decay.

7. ACKNOWLEDGMENTS

We thank Masayuki Ikeda, Takumi Maruyama, Akira Katsuno and Mariko Sakamoto who gave us many advices. This research was supported in part by the Grant-in-Aid for Creative Basic Research, 14GS0218, 17680005, and PRESTO, Japan Science and Technology Agency.

8. REFERENCES

[1] CACTI, http://research.compaq.com/wrl/people/jouppi/CACTI.html

[2] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," *Proc. of the 29th Int. Symp. on Computer Architecture*, pp.148-157, May 2002.

[3] R. Fujioka, K. Katayama, R. Kobayashi, H. Ando, and T. Shimada, "A Preactivating Mechanism for a VT-CMOS Cache using Address Prediction," *Proc. of the Int. Symp. on Low Power Electronics and Design*, pp.247-250, Aug. 2002.

[4] S. Heo, K. Barr, M. Hampton, and K. Asanovic, "Dynamic Fine-Grain Leakage Reduction Using Leakage-Biased Bitlines," *Int. Symp. on Computer Architecture (ISCA29)*, pp.137-147, May 2002.

[5] S.Kaxiras, Z.Hu, and M.Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," *Proc. of the 28th Int. Symp. on Computer Architecture*, pp.240-251, June 2001.

[6] C. H. Kim and K. Roy, "Dynamic Vt SRAM: A Leakage Tolerant Cache Memory for Low Voltage Microprocessors," *Int. Symp. on Low Power Electronics and Design (ISLPED2002)*, pp.251-254, Aug. 2002.

[7] N.S.Kim, K.Flautner, D.Blaauw, and T.Mudge, "Drowsy Instruction Caches; Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction," *Proc. of the Int. Symp. on Microarchitecture*, pp.219-230, Nov. 2002.

[8] L.Li, I.Kadayif, Y-F.Tsai, N.Vijaykrishnan, M.J.Irwin, and A.Sivasubramaniam, "Leakage Energy Management in Cache Hierarchies," *Proc. of the 11th Int. Conf. on Parallel Architectures and Compilation Techniques*, pp.131-140, Sep.2002.

[9] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, "Soft Error and Energy Consumption Interactions: A Data Cache Perspective," *Proc. of the Int. Symp. on Low Power Electronics and Design (ISLPED'04)*, pp.132-137, Aug. 2004.

[10] D.Parikh, Y.Zhang, K.Sankaranarayanan, K.Skadron, and M.Stan, "Comparison of State-Preserving Leakage Control in Caches", *2003 Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, June 2003

[11] M. Powell, S. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," *Int. Symp. on Low Power Electronic and Design*, pp.90-95, July 2000.

[12] A. Sakanaka and T. Sato, "A Leakage-Energy-Reduction Technique for High-Associativity Caches in Embedded Systems," *Workshop on Memory Access Decoupled Architectures and Related Issues*, pp.51-56, Sep. 2003.

[13] SimpleScalarLLC, http://www.simplescalar.com

[14] SPEC –Standard Performance Evaluation Corporation, http://www.spec.org/

[15] S.H.Yang, M.D.Powell, B.Falsafi, K.Roy, and T.N.Vijaykumar, "An Integrated Circuit / Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches," *Proc. of the 7th Int. Symp. on High-Performance Computer Architecture*, pp.147-157, Feb.2001.

[16] S.H.Yang, M.D.Powell, B.Falsafi, and T.N.Vijaykumar, "Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay," *Proc. of the 8th Int. Symp. on High-Performance Computer Architecture*, pp.151-161, Feb.2002.