Quantitative Evaluation of Leakage Reduction Algorithm for L1 Data Caches

Komiya, Reiko
Institute of Systems & Information Technologies /KYUSHU | Dept. of Elec. Eng. and Computer Science Fukuoka University

Inoue, Koji
Dept. of Elec. Eng. and Computer Science Fukuoka University

Moshnyaga, Vasily G.
Dept. of Elec. Eng. and Computer Science Fukuoka University

Murakami, Kazuaki
Dept. of Informatics Kyushu University | Institute of Systems & Information Technologies | KYUSHU

https://doi.org/10.15017/6168

出版情報：Proc. of the The International SoC Design Conference (ISOCC04), pp.582-585, 2004-10. The International SoC Design Conference
バージョン：
権利関係：

Quantitative Evaluation of Leakage Reduction Algorithm for L1 Data Caches

R. Komiya[1,2] komiya@v.tl.fukuoka-u.ac.jp
K. Inoue[2] inoue@tl.fukuoka-u.ac.jp
V. G. Moshnyaga[2] vasily@fukuoka-u.ac.jp
K. Murakami[1,3] murakami@i.kyushu-u.ac.jp

[1] Institute of Systems & Information Technologies /KYUSHU Fukuoka, Japan
[2] Dept. of Elec. Eng. and Computer Science Fukuoka University, Fukuoka, Japan
[3] Dept. of Informatics Kyushu University Fukuoka, Japan

Abstract - ***A number of techniques to reduce cache leakage have so far been proposed. However, it is not clear that 1) what kind of algorithm can be considered and 2) how much they have impact on energy and performance. To answer the questions, this paper classifies cache-leakage reduction techniques and evaluates their energy-performance efficiency. As a result, we have found that an approach employed by the Drowsy cache [1] achieves the best energy-performance efficiency with low complexity. Moreover, we investigate the potential of the approach on multi-thread program executions.***

Keywords: Low Power, cache, Leakage.

1 Introduction

As the popularization of battery operated devices, energy consumption has become an inevitable issue for microprocessor designs. Energy consumption of CMOS circuits consists of two factors: dynamic energy and static (or leakage) energy. Although the former dominates the total amount of energy, the impact of the leakage has been increasing rapidly with the decrease of feature size. In state-of-the-art microprocessors, on-chip caches are major contributors to the leakage, because the amount of leakage energy strongly depends on the number of transistors to be employed.

To challenge the leakage problem, a number of researches have proposed reduction techniques [1][2][4][5]. A common concept of these techniques is to support two operation modes; high-leakage but high-speed mode (or *awake mode*) and low-leakage but low-speed mode (or *sleep mode*). Then the main difference is how to select an appropriate operation mode in order to achieve low-leakage energy without any performance degradation. Since the leakage reduction algorithm gives great impact on both energy and performance, analyzing and comparing the potential of mode control strategy are very worthwhile.

In this paper, we focus on the cache management algorithms and classify them to explore the design alternatives of low leakage caches. Moreover, based on the classification, we evaluate energy-performance efficiency on single- and multi-thread executions. For fair evaluations, an assumption is used throughout the paper, line by line fine-grain optimization with state-preserving approach. Namely, we can manage the operation mode of each cache line independently and the contents of cache line are retained even in the sleep mode.

This paper is organized as follows. Section 2 classifies cache-leakage reduction algorithms. In Section 3, we evaluate energy-performance efficiency of some cache-management strategies. Finally, Section 4 concludes this paper.

2 Classifying Leakage Reduction Algorithms

If the cache aggressively chooses the sleep mode, large amount of leakage can be reduced. However, in this case, the cache may have negative influence on performance because accessing to sleep lines increases cache access time. In this section, we classify and explore the mode control strategy for low leakage caches.

2.1 Switching to the Sleep Mode

Ideally, a cache line which is loaded from the next level memory should be turned off just after its last access. In order to approach to the ideal management, low leakage caches predict whether or not each cache line is to be reused. Table 1 summarizes the algorithm to change the cache-line state to the sleep mode. The table also presents the relation with previously proposed techniques [1][2] [4][5]. We can consider at least the following three design decisions for sleep-mode algorithm.

- ***What is a suitable condition?***: When the condition is satisfied, the corresponding line is switched to the sleep mode. As shown in Table 1, there are at least two types of condition: counter type and event type. The counter type counts the number of target events. The condition is satisfied if the value of counter gets equal to or smaller (or larger) than a threshold. For example, when the counter of cache misses gets a smaller value than a given threshold, we consider that the current cache size is too large for the target program. In this case, the cache should aggressively select the sleep mode. Another example is that counting the number of

Table 1 : Mode Transition Algorithm the Sleep Mode

Condition		Period	Validation
type	counter ≦ threshold (cache misses, accesses)	TW	syncronous
counter (target)	counter ≧ threshold (execution cycles[2])	TW[2]	syncronous[2]
	counter ≧ threshold (cache hits, no-access cycles[1])	TW	syncronous
			asyncronous
		NTW[1]	- - -
event	Load issue[3]	NTW[3]	- - -

[1] Cache Decay [2] Drowsy cache [3] Cache Hierarchy

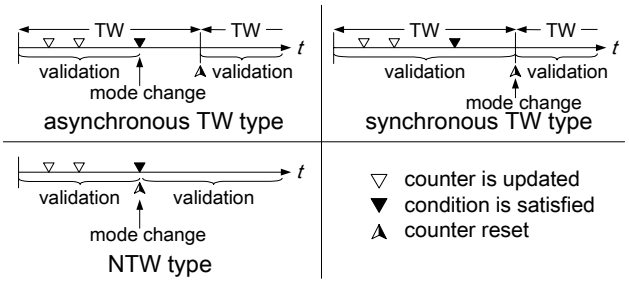

Figure 1 : Mode Transition Example

accesses (or non-access cycles) can be used to capture the amount of temporal locality of memory references. The simplest approach is to count just only the execution cycles, in which the each cache line enters to the sleep mode at every fixed cycle elapsed. On the other hand, the event type does not consider previous history of events, thus the condition is satisfied just when the event occurs.

- ***How long should the condition be monitored?***: The period to monitor the mode control condition effects leakage reduction efficiency. A short-period monitoring can correspond to quick transition of memory access patterns, but there is a possibility to make hasty decisions. We have two options for the monitoring period: time-window (TW) and no-time-window (NTW). The TW monitors the condition during a fixed time interval, while the NTW does not have any time period. In the case of counter type, it is necessary to reset the value of counter to zero. The TW scheme initializes each counter at the end of time window. On the other hand, the NTW scheme resets the counter just when the condition is satisfied. Table 1 shows the possible combinations with the mode-change conditions.

- ***When is the condition validated?***: For the NTW scheme, the cache transits its operation mode just when the condition is satisfied. For the TW type, we can consider two options: **synchronous** and **asynchronous**. The former switches the operation mode at the end of the TW, i.e. the cache waits for validating the conditional decision until the end of the current TW, while the later changes the operation mode soon when the condition is satisfied. The advantage of the asynchronous strategy is that it may reduce leakage energy effectively due to quick response, but it is required to check the value of counter at every counter up-dating. On the other hand, the frequency of conditional checking is lowered, but the synchronous approach causes some delay to enter the sleep mode.

Table 2 : Mode Transition Algorithm to the Awake Mode

Condition		Period	Validation
type	counter ≦ threshold (cache hits, no-access cycles)	TW	syncronous
counter (target)	counter ≧ threshold (execution cycles)	TW	syncronous
	counter ≧ threshold (cache misses, accesses)	TW	syncronous
			asyncronous
		NTW	- - -
event	Store issue[3]	NTW [1][2][3]	- - -
	Load/Store issue[1][2]		

[1] Cache Decay [2] Drowsy cache [3] Cache Hierarchy

Figure 1 shows examples for the combination of the above three decisions. In this figure, the counter is up-dated two times (as marked by the white rectangles) until the condition is satisfied (marked by the black rectangle). In the case of the asynchronous TW type, the mode is switched when the condition is satisfied and the counter is reset to zero at the end of TW. On the other hand, the synchronous TW performs mode transition and counter initialization simultaneously at the end of TW. Unlike them, in the NTW type, the mode changing and the counter resetting are done when the condition is satisfied.

2.2 Switching to the Awake Mode

The classification of the mode switching algorithm from sleep mode to awake mode is shown in Table 2. This algorithm has complementary relation with sleep mode algorithm except that condition which counts execution cycles. Moreover, in order to take the locality of memory references into account, we add “Load or Store issue” in event type. Unlike Table 1, many leakage reduction techniques concentrate on the event type. This is because the temporal locality of memory references is applied. The case of event type, the fall of performance is suppressed. But counter type approaches need to keep counting the number of events during a monitoring period.

3 Evaluation

Based on the classification discussed in Section 2, we evaluate the energy-performance efficiency of leakage reduction algorithms.

3.1 Leakage Energy Model

In this evaluation, we approximate total leakage energy, *LEtotal*, consumed in a L1 data cache as follows:

Table 3 : Benchmark Program Patterns by MT

	Max	Mid	Min
LE	i253.perlbmp i254.gap f200.sixtrack i256.bzip2	f188.ammp i186.crafty i255.vortex i197.parser	f179.art i176.gcc i181.mcf f178.galgel
ET	i176.gcc i164.gzip f179.art f173.applu	f191.fma3d i255.vortex i252.eon i175.vpr	i253.perlbmp i254.gap f168.wupwise f200.sixtrack

$$LEtotal = CC * LEline * Nline \quad (1)$$
$$CC = CCconv + CCextra \quad (2)$$
$$LEline = SR * LEsline + (1-SR) * LEaline \quad (3)$$

Where *CC* is the total number of execution cycles, *LEline* is the average leakage energy consumed in a cache line per cycle, and *Nline* is the number of lines in the L1 data cache. *CC* consists of *CCconv* and *CCextra*. *CCconv* is the total execution time without any leakage reduction techniques, while *CCextra* is the penalty caused by accessing to lines in the sleep mode. *LEline* is determined by *SR* which is a rate of sleep mode lines in the entire cache. *LEsline* and *LEaline* are the cache-line leakage energy dissipated in one cycle on the sleep and the awake mode, respectively. Based on [1], we assume that the ratio of *LEaline* and *LEsline* is 100 to 8. This result was obtained by performing circuit-level simulation with a 0.07μ*m* Berkeley Spice model.

3.2 Experimental Environment

We measured *CC* and *SR* by performing trace-driven simulations. For this evaluation, we used the ZonC SPARC64 simulator [3], and executed 26 programs from the SPEC2000 benchmark for single-thread executions and 6 combinations of 4 programs for multi-thread executions. Since each program is completely independent, for the multi-thread evaluation, we can obtain high instruction-level parallelism. Table 3 summarizes the detail of program combinations, each of which is determined based on leakage-energy (LE) and execution-time (ET) on the Drowsy [1] management algorithm. For example, the best 4 programs for leakage reduction with the Drowsy strategy are grouped into the LE-Max, and the worst 4 programs belong to LE-Min. Similarly, ET-Min is a combination of programs which produce minimum impact on performance.

Based on a SPARC64 design, 128 KB two-way set-associative data L1 cache with a 64B line size is assumed, thus *Nline* in equation (1) is 2K. In this simulation, the first 50 million instructions are skipped for warming-up the program execution and the following 10 million instructions are used for measurements. To evaluate the potential of each algorithm, we ignore the dynamic energy consumption for cache accesses and mode-control circuits. Furthermore, we introduce the following assumptions.

Table 4 : Evaluated Mode

Model	To Sleep Mode			To Awake Mode		
	Cond.	Peri.	Valid.	Cond.	Peri.	Valid.
EC-EC	EC	TW	Sync.	EC	TW	Sync.
EC-NAC	EC	TW	Sync.	NAC	TW	Sync.
EC-S	EC	TW	Sync.	S	NTW	- - -
EC-LS	EC	TW	Sync.	LS	NTW	- - -
NAC-EC	NAC	NTW	- - -	EC	TW	Sync.
NAC-NAC	NAC	NTW	- - -	NAC	TW	Sync.
NAC-S	NAC	NTW	- - -	S	NTW	- - -
NAC-LS	NAC	NTW	- - -	LS	NTW	- - -

EC:Execution Cycles NAC:No-Access-Cycle
S:Store LS:Load/Store

- It is allowed to access to sleeping cache lines without switching to awake mode. However, in this case, some penalty on cache access is caused. We call the overhead Sleep-Hit-Penalty (SHP).
- Regardless of the algorithm, the line loaded from the next level memory is initially set to the awake mode.
- The tags are always in the awake mode, because sleeping tags degrades significantly the performance in spite of relatively low leakage reduction [1].

The cache models to be evaluated are shown in Table 3. Here, we choose two counter-type conditions for the sleep algorithm, one counts execution cycles (EC) and the other counts no-access cycles (NAC), because they are representative conditions in line-base leakage optimization techniques[1][2]. On the other hand, for the algorithm to wake lines up, two of event-type conditions, load (L) and load/store (LS), and two of counter-type conditions, EC and NAC, are considered. For the model notation, the left and right characters indicate the condition to enter the sleep and awake mode, respectively. We assumed that the threshold to satisfy the conditions is set to 4K cycles based on the reference [1].

3.3 Results

Figure 2 shows the leakage reduction and performance overhead compared with a non-optimized conventional cache. First, we discuss the effects of wake up strategy. Regardless of the sleep algorithm employed, the event type approaches (L and LS) achieve significant leakage reduction compared to the counter type models, EC and NAC. This is because the counter type approach wakes up all of the cache lines at every fixed interval without any consideration for access behavior. As a result, a number of lines whose last access has already been completed are woken up, degrading the efficiency of leakage reduction. On the other hand, a mode transition takes place only when a load or store instruction is issued in the event type approaches, so that the majority of cache lines keep to residing in the sleep mode. Moreover, we see from Figure 2 that the leakage reduction rates of EC-S and EC-LS are comparable and this situation can also be seen for NAC-S and NAC-LS models. However, the store-event wake-up

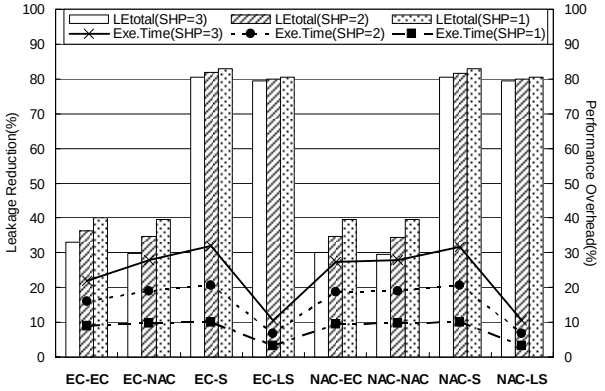

Figure 2 : Results on Single-Thread Executions (Average of all benchmarks)

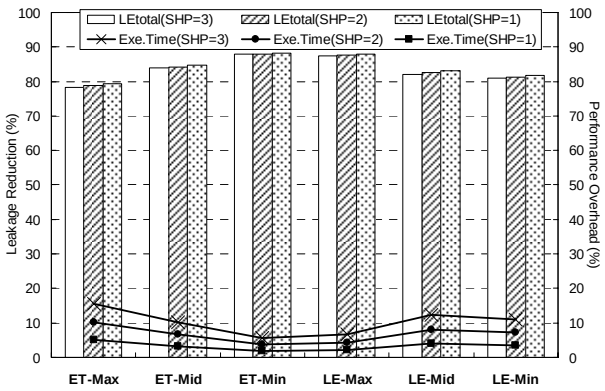

Figure 3 : Results on Multi-Thread Executions (EC-LS)

algorithm produces much larger performance degradation than the load/store wake-up approaches. The result comes from the fact that memory references have temporal and spatial locality regardless of the access operations, load or store. This negative effect becomes clear with increase in the SHP. If the SHP is three clock cycles, the performance is degraded by about 30% on EC-S and NAC-S.

Next, we consider the algorithm to make cache lines enter to the sleep mode. Here, based on the results discussed above, we focus only on the two models employing event-type wake-up strategy, EC-LS and NAC-LS. From the simulation results, we can not see large differences for energy and performance. However, EC-LS has an advantage over the NAC-LS for the complexity of implementation. Since NAC-LS requires monitoring whether or not each cache line is accessed. On the other hand, EC-LS does not consider the cache-access behavior, and counts just the number of clock cycles elapsed.

From the results discussed above, we conclude that EC-LS approach which is employed by the drowsy cache is the most promising algorithm to achieve low-leakage L1 data caches.

Finally, we discuss the energy-performance efficiency on multi-thread execution. Here, we focus only on the best strategy, EC-LS, due to the lack of space. In multi-thread executions, the amount of locality of memory references will be reduced due to distributed accesses from different threads. However, as shown in Figure 3, EC-LS can maintain high reduction rates as well as the single-thread executions. Therefore, we conclude that EC-LS is a promising approach even on multi-thread high-performance processors.

4 Conclusions

In this paper, run-time cache management algorithm to reduce leakage energy consumption has been classified. In addition, we have evaluated energy-performance efficiency. As a result, we have found that "EC-LS" algorithm achieves the best energy-performance efficiency with relatively low complexity. This model makes each cache line switch to the sleep mode at every fixed interval and sleeping line are transited to the awake mode when they are accessed. In this evaluation, the dynamic energy overhead caused by mode control units is not contained. To evaluate with more accurate energy model including not only static energy but also dynamic energy consumed for the run-time cache management is future works.

References

[1] 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.

[2] 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.

[3] M.Sakamoto, A.Katsuno, A.Inoue, T.Asakawa, H.Ueno, K.Morita, Y.Kimura, "Microarchitecture and Performance Analysis of a SPARC-V9 Microprocessor for Enterprise Server Systems," *Proc. of the 9th Int. Symp. on High-Performance Computer Architecture*, pp.141-152, Feb.2003

[4] 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.

[5] 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.