Reducing leakage in power-saving capable caches for embedded systems by using a filter cache

Roberto Giorgi
University of Siena
Via Roma, 56
53100 Siena - Italy
+39 0577 234630
www.dii.unisi.it

Paolo Bennati
University of Siena
Via Roma, 56
53100 Siena - Italy
+39 0577 233601
www.dii.unisi.it

ABSTRACT

Leakage power in data cache memories represents a sizable fraction of total power consumption, and many techniques have been proposed to reduce it. As a matter of fact, during a fixed period of time, only a small subset of cache lines is used. Previous techniques put unused lines, for example, to drowsy state or switch them off completely (cache decay) in order to save power.

Our idea is to adaptively select mostly used cache lines. We found that this can be achieved automatically by using a tiny cache acting as a filter “L0” cache. Our main contributions are: i) evaluation of filter cache to reduce leakage; ii) improvement of other existing power-saving techniques; iii) providing results to select the most promising solution.

Our experiments, with complete MiBench suite for ARM based processor, show (in average) 10% improvement in leakage saving and 17% in leakage energy-delay versus drowsy-cache; versus decay-cache we found 6% improvement in leakage saving and 13% in leakage energy-delay.

Categories and Subject Descriptors

B.3.2 [**Memory structures**]: Design styles – *cache memories.*

General Terms

Performance, Design, Experimentation.

Keywords

Cache decay; drowsy cache; filter cache; low-power; leakage.

1. INTRODUCTION

Power consumption of cache memories represents a large fraction of the total power consumption; previous studies have found that it accounts for about 50% of the total power consumed in embedded computing systems [1, 2].

The causes for power consumption in cache memories are mainly due to two factors: *dynamic power* and *static power*. Dynamic switching power is due to the charging and discharging of parasitic and/or input capacitors. Static power, which is due to sub-threshold, gate-oxide and reverse biased PN junction leakage, has increased in importance in recent CMOS technologies. Leakage depends mainly on the number of transistors and their features and, as also predicted in many studies [3-6], from the 70nm generation, it constitutes a large part of total power dissipation.

Recently, many research projects have focused on reducing leakage power in the cache memories [2, 4, 5, 7-15]. The common idea is to put unused cache lines into a power-saving state. The proposals can be broken down into two main categories [15]: *state-preserving* and *non-state-preserving* techniques. When a cache line is put in a power-saving state, the technique is called state-preserving if the line content is maintained (although accessible only after putting it back in normal state) and, on the other hand, non-state preserving if it is destroyed. State preserving techniques are mainly based on the circuit technique called *drowsy* [5], while non state-preserving techniques utilize the circuit technique called *gated-Vdd* [6]. The first has the advantage that no data is lost (only a wake-up is needed), but the leakage saving is less than in the second one; the second one provides a high leakage saving, but unfortunately, data stored in lines are lost and, in case of cache miss, the cell has to be turned on with a consequent performance loss.

Both techniques work well if the selection of the lines to put in power-saving mode is done accurately. It is important to carefully select *which* lines to deactivate and *when*. This is necessary to avoid performance loss and to achieve enough leakage saving. As a matter of fact, during a fixed period of time, only a small subset of cache lines is used [16], so there are a lot of possibilities for saving power.

In [17], the use of a L0 cache (*filter cache*), that is very small relative to the conventional L1 cache, has been proposed to reduce L1 cache activity. The filter cache works as a buffer to cache recently accessed cache lines. This approach reduces the activity of the L1 cache.

In this paper, we propose to reduce the time when cache lines is actively dissipating power. Initially, we considered several options to shorten this time by considering, for example, to dynamically switch between drowsy and decay, as proposed by Meng et al. [18]. Our idea here is to make lines in L1 “less used” by using an L0 filter cache. This solution is very effective for the following reasons: i) an L0 cache can reduce the average access time as a small cache is faster the a larger L1 power-saving cache; ii) the additional cost of powering-up an L0 cache is much lower than the power-saving obtained by the faster powering-down of less frequently used L1 lines (as showed by experiments throughout the paper); iii) the additional design efforts to include an L0 cache is conceptually simple.

Power-saving caches are typically used at the first level in the hierarchy. Our findings demonstrate that not only a speed-up can be achieved by using another small cache level (the total amount of cache memory is increased) but it also provides benefits in power-saving. This is due to the fact that the filter cache introduces itself a negligible leakage consumption (it is very tiny relative to the lower level) and the leakage in the conventional power-saving cache at the lower level is reduced because that level works for a smaller period of time. Indeed, the ratio of lines that are in power-saving state is not reduced and the number of transitions between the two level (high-power to low-power and viceversa) is smaller.

Substantially we show that the addition of a filter cache before conventional power-saving caches provides advantages for both leakage-saving and performance (the IPC increments).

The remainder of the paper is organized as follows. In the next section we give our motivation for introducing L0. In Section 3 the details of our proposal are presented. In Section 4, we explain the methodology we used for our experiments. Then we discuss, in Section 5, the results we have obtained, we evaluate the additional leakage saving achieved and the performance improvements versus other literature proposals for power-saving caches. Section 6 illustrates some recent related works. We finally conclude in Section 7.

2. MOTIVATION

In order to motivate our investigation, we analyzed the results with respect to the cases that can be considered as ideal theoretical cases.

For a standard drowsy-cache, the best theoretical situation that can be achieved is when during each cycle only a line (the currently accessed line) is in high-power state while the others are in drowsy state. This ideal case could be obtained with the perfect knowledge of the access pattern, so that the power-saving technique is able to wake up the line in time and there is no possibility of accessing a drowsy line. We refer to this case as *Ideal drowsy* (versus *Drowsy*). An analogue situation can be considered the ideal theoretical case for decay-cache. Ideally, if the access pattern is perfectly known, only one line (the currently accessed one) could be kept on, while all the others are completely gated-off. We refer to this case as *Ideal decay* (versus *Decay*). We consider as ideal case a situation where just the currently accessed line is on (maximum saving) and the next line to be accessed is available immediately (no delay). This hypothesis is really very near to the ideal situation (more ideal, for example, than the oracle predictor presented in [19]).

The behaviour for the leakage-saving (how much leakage can be saved by using the power-saving technique) of each approach, compared to the ideal cases, is shown in Figure 1.

Both drowsy and decay techniques are effective for leakage saving, but there is a lot of distance that can be filled between the real and the ideal case. There is about a 10% of additional leakage saving that can be achieved and we are proposing a solution that try to reduce this distance. Looking at the performance, drowsy cache almost doesn't impact the IPC (there is a reduction less than 0.5%), while the performance loss for decay cache is higher (about 20%).

In order to clearly understand the effectiveness of the techniques for both power-saving and performance, a metric that is able to combine them together is useful. We evaluate the leakage energy-delay product in a way that is appropriate for our study [20]. Since we are interested in the leakage, we calculated the product as: *leakage energy * execution time* (Figure 2). With this metric, we can see that drowsy and decay are almost equivalent: decay provides better leakage saving while drowsy produces less performance loss, but considering these two factors together the techniques are very similar. However they are both far from their theoretical ideal case (more than 10% of difference).

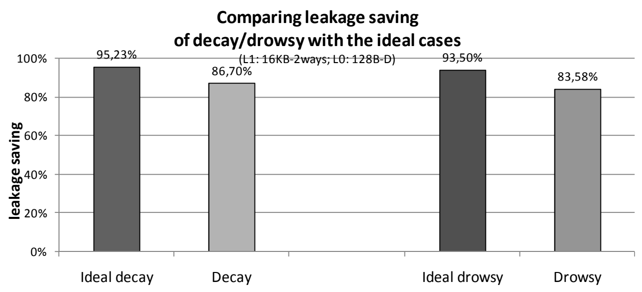


Figure 1. Comparison of leakage saving of decay/drowsy with theoretical ideal cases (full Mibench suite average).

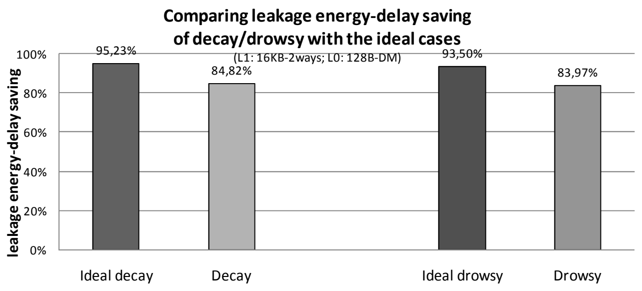


Figure 2. Comparison of leakage energy-delay saving of decay/drowsy with theoretical ideal cases (full Mibench suite average).

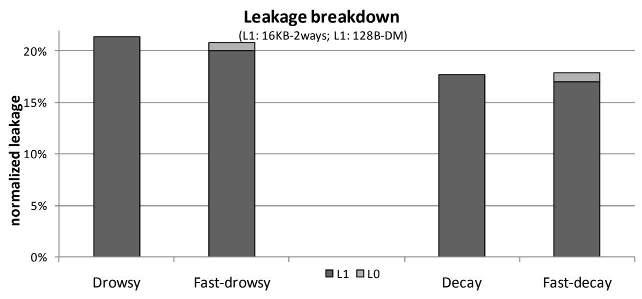


Figure 3. Leakage breakdown for tiffdither. The baseline (100%) is the leakage spent in standard cache (L1 cache without power-saving technique).

Figure 3 shows the leakage breakdown. A benchmark is presented as a case study. As shown, the additional leakage introduced by L0 is negligible. It accounts for about 1% of total leakage. This is actually the extra leakage in which our proposal incurs. However, since the reduction in L1 leakage is considerable, the total leakage (the sum of the leakage in L0 and in L1), is reduced.

Results presented in this section show that there are a lot of possibilities for saving more power without penalizing performance.

3. FAST POWER-SAVING CACHE HIERARCHY

We considered an embedded system architecture based on an ARM processors and in this scenario our proposal addresses the problem of low-power data cache memories. A simple solution that has not been evaluated, as of our knowledge, for reducing leakage is based on a filter cache placed between CPU and a convention first level cache with power-saving capabilities (drowsy/decay). We call this scheme *fast-drowsy/decay*.

Many different approaches can be used to reduce leakage in cache [2, 4, 5, 7-15]. We have focused on the following two techniques: drowsy cache [5] for the state-preserving category, and cache decay [19] for the non-state-preserving one.

L0 cache is a very tiny (e.g. 128B) and fast cache [17], therefore the latency between L0 and CPU could be, for example, 1 cycle. We considered a typical size for an L1 power-saving cache for embedded systems (e.g. 16 KB) Due to the power saving techniques and larger size, this cache can experience a longer latency (e.g. 2 cycles).

Our proposal aims to reduce leakage while not reducing performance. The addition of a new level of cache provides faster execution, but we show in this paper that this solution has many benefits also when the concern is the leakage saving.

In order to understand the improvement we propose, we present Figure 4; it shows in detail the transitions between high-power and low-power states, comparing our proposal versus the conventional scheme. Power-saving techniques can select the lines to put into the low-power state by using many different policies [5, 12, 13]. The simplest policy that can be considered is one where, periodically all the lines in the cache are put into low-power mode and a line is waken up only when it is accessed again. This policy is called *Simple policy* and it requires only a single global counter as additional hardware (apart from the circuitry for the voltage reduction).

This behaviour (conventional scheme) is shown in Figure 4 (a): the power-saving technique with simple policy [5] put all the lines into the low-power state (e.g. drowsy or decay state) when the global counter saturates. At the next access, the accessed line must be waken up. This translates into a performance loss and also into a dynamic power cost.

Figure 4 (b) shows how our proposal (fast scheme) acts. We propose to avoid to uselessly waking-up of lines by adding a filter "L0" cache before the L1 power-saving capable cache. L0 cache captures the most recently used accesses. Lines can remain off in L1 since the last used data are stored also in L0 and an hit in the filter cache solves the access. Being L0 small and fast it introduces negligible additional leakage.

The main contributions of the addition of L0 are: i) lines get older faster (accesses can be solved in L0 without waking-up lines again in L1); ii) the number of switching between low-power and high-power state are reduced (with the consequent reduction of the related dynamic power).

This unexplored solution has several advantages:

- It mitigates the IPC degradation power-saving caches suffer;
- It requires just simple SRAM cache;
- It consumes a negligible fraction of a low-power L1 cache (e.g. drowsy or decay);
- It adapts to programs behaviour without any special effort.

L0 adapts to programs behaviour without any special effort, providing an automatic static filtering operation for the accesses into L1; if a program needs more than the size of L0 allows, the L1 will work as a conventional power-saving cache, otherwise, L1 can reduce the switching between low-leakage state and high-power state and it can maintain lines into power-saving state for a longer time.

The additional cost of this solution can be less than 1% of the L1 cost. The size of L0 is considerably small and it is negligible relative to the full size of the data cache memory hierarchy. In addition, L0 doesn't have any additional circuitry for power-saving (such as additional bit, voltage supply, control).

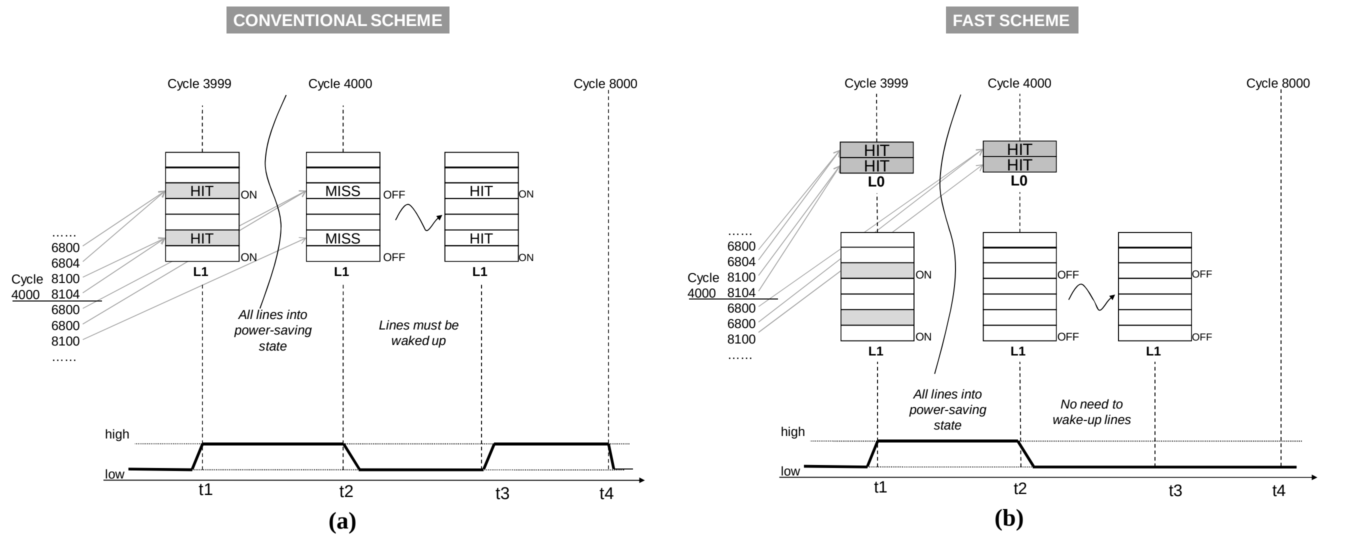


Figure 4. Comparison between power-saving schemes. (a) Conventional power-saving scheme with Simple-policy; (b) Fast power-saving scheme (our proposal). Two assumptions have been done: i) global counter saturates every 4000 cycles; ii) block size is 8kbytes.

Substantially our technique provides a faster program execution maintaining the same ratio of line in power-saving state in L1. Since the activity time is reduced, the total leakage consumption is reduced. The overhead of the additions we propose is negligible (L0 is very small and it consumes very poor leakage), so the total leakage is reduced, as we are going to show in the experiments presented in §5.

4. SIMULATION SETUP

All simulations have been performed with HotLeakage [21] simulator, as in [4, 8, 11, 13, 15, 18, 22] retargeted for ARM based processor and modified in order to implement our configuration. For technology parameters, we use values for a 70nm process with Vdd=0.9V (Table 1) as presented in several current studies [13] [15]. This choice can provide a more direct comparison with such studies.

In this work we analyze a fixed configuration for the power-saving technique (simple policy with 4000 cycles update window).

We simulated the entire suite of MiBench [23] benchmark suite for an ARM based processor. We compared conventional leakage power-saving caches with the scheme we propose and we also compared them with a standard cache (without power-saving capabilities) in order to have a common baseline. To be more clear, let consider the following acronyms:

- **standard cache**: level 1 (L1) data cache *without* filter cache (L0); no power-saving technique applied;
- **drowsy cache**: level 1 (L1) data cache *without* filter cache (L0); *drowsy cache* technique applied to L1 cache;
- **decay cache**: level 1 (L1) data cache *without* filter cache (L0); *decay cache* technique applied to L1 cache;
- **fast-drowsy cache**: level 1 (L1) data cache *with* filter cache (L0); *drowsy cache* technique applied to L1 cache;
- **fast-decay cache**: level 1 (L1) data cache *with* filter cache (L0); *decay cache* technique applied to L1 cache;

Table 2 shows the configuration of the caches we simulated. L0 cache size is closest its typical size [17, 24, 25] and the other parameters are closest to common ARM XScale processor [26].

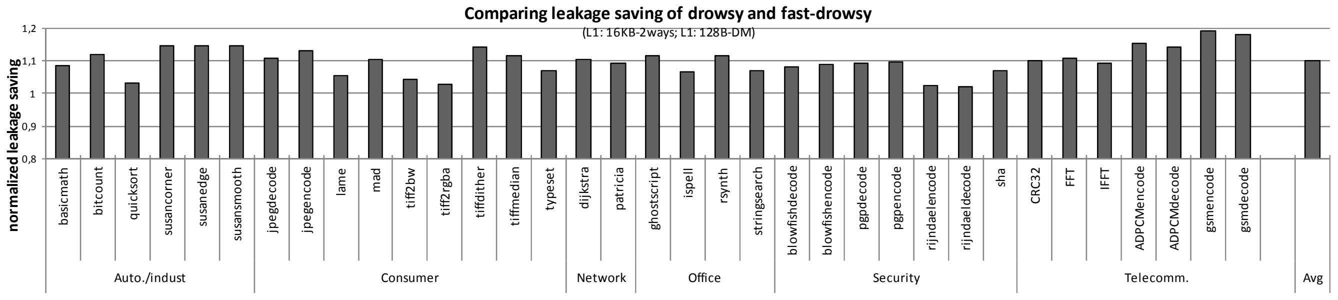
For the leakage evaluation, we take into account all the extra power consumptions, in particular the dynamic energy, that each power-saving technique introduces; in other terms, we consider the total leakage, where we account for the contributions from activity in the counters the power-saving techniques use to periodically put lines into low-power state, the leakage of extra circuitry and we consider also the dynamic power of such extra circuitry (as an additional cost to implement the low-power technique). The leakage, for the configuration where L0 is included, accounts for the L0 contribution: the total leakage considered is the sum of the leakage spent in L0 and the leakage spent in L1.

Table 1: Energy parameters

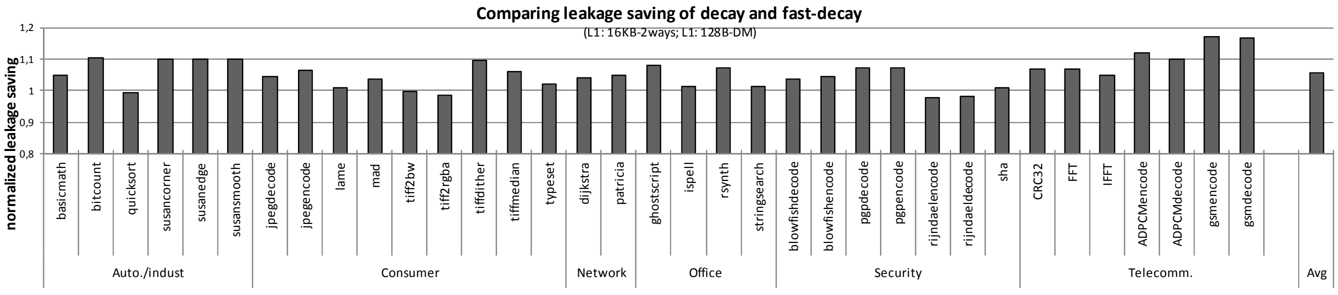
Leakage control technique	**Drowsy**	**Decay**
Time for low to high switch	3 cycles	3 cycles
Time for high to low switch	3 cycles	30 cycles
Low to high switch cost	0.0003 nJ	0.0003 nJ
High to low switch cost	0.0001 nJ	0.0001 nJ
Extra latency in low leak mode	1 cycle	0 cycle
Policy	Simple	
Update window	4000 cycles	

Table 2: Configuration of the caches

	L0	**L1**
Cache size	128B	16KB
Block size	16B	16B
Associativity	Direct mapped	2
Latency	1 cycle	2 cycles



(a)



(b)

Figure 5. Leakage-saving across MiBench suite: higher is better. (a) Comparison of leakage-saving in fast-drowsy and drowsy: baseline is the leakage-saving achieved with drowsy (drowsy itself in average produces a leakage-saving of 83.58% relative to the case.

5. RESULTS

The main objective of this work is to reduce leakage in data cache memories without penalizing performance. In order to demonstrate the effectiveness of our proposal, we studied mainly three metrics: leakage, IPC and energy-delay product (as introduced in § 2.1). In the following, these metrics are analyzed in detail and the results obtained across MiBench suite [23] are presented.

5.1 Leakage saving

Figure 5 shows the leakage saving achievable with our proposal in comparison with standard techniques. Figures 4, 6, 7 show the behaviour of our proposal across all the benchmarks of the MiBench suite [23].

Figure 5 (a) shows the comparison of the leakage saving obtained with fast-drowsy and with a drowsy-cache (used as baseline). As shown, fast-drowsy is always better than drowsy-cache and the behaviour is almost regular. The reduction in leakage (or improvement in leakage-saving) varies from a minimum of 2% to a maximum of 19% and in average, fast-drowsy reduces the leakage, in comparison with drowsy-cache, of about 10%.

The same results for fast-decay are presented in Figure 5 (b) in comparison to the decay-cache. For this technique the benefits for leakage saving are less significant. This is not surprising, because the decay technique itself already provides a very powerful leakage saving, higher than drowsy. For a few benchmarks (quicksort, tiff2bw, tiff2rgba and rijndael) fast-decay is even worse, even though the loss is minimal (less than 2%). The main problem for these benchmarks should be the size of L1 because on it there is a huge amount of conflict miss that can't be eliminated. However, preliminary simulations with a bigger cache (e.g. 64KB) show this problem can be overcome and that fast-decay should produces benefits some benefits also for them. In average, fast-decay reduces the leakage of about 6% in comparison to decay-cache.

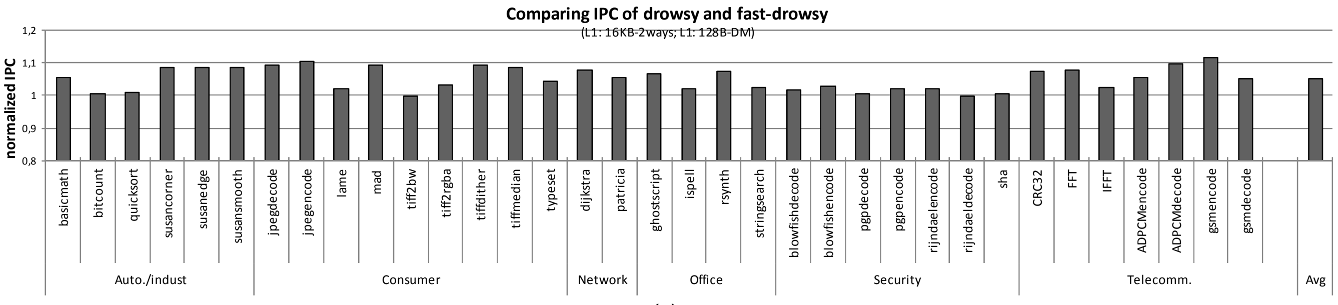
If the leakage spent by a standard cache (without power-saving technique) is assumed as baseline (it is not shown in figures) drowsy-cache reduces leakage to 16.4% while fast-drowsy to 14.7% (in average) and decay-cache reduces leakage to 13.3% while fast-decay 12.8%. Anyway, it is more appropriate to compare this solutions in terms of leakage energy-delay as discussed below.

5.2 IPC

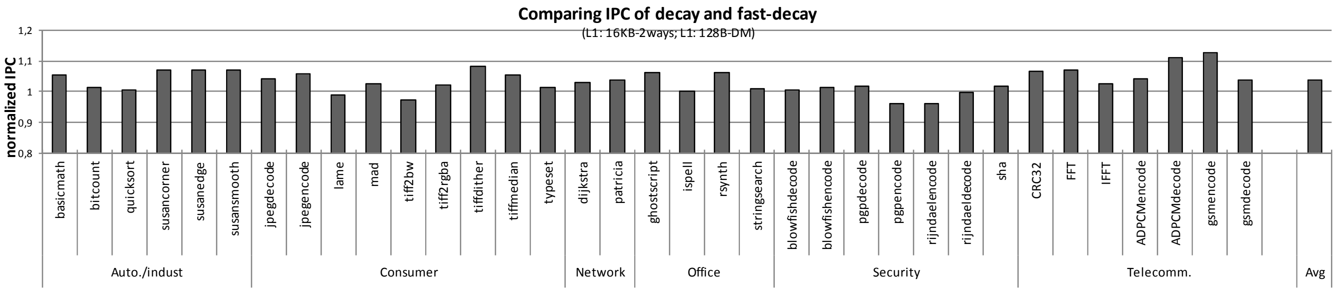
We wish to adopt a solution that at the same time guarantees performance and power-saving capabilities. With this assumption, in order to demonstrate how our solution addresses this challenge, we investigated the performance in terms of number of instruction executed during a cycle (IPC). The filter cache, in average, improves the IPC both for drowsy cache and decay cache. It provides faster access to the mostly used data and this increments the number of instruction executed during a cycle.

Figure 6 (a) shows results for drowsy technique. The IPC for fast-drowsy is compared to the IPC for drowsy-cache (used as baseline). For each benchmark, the filter cache increases the IPC. The drowsy technique itself doesn't suffered too much of performance loss, but with the addition of a tiny filter cache, an additional speed-up is possible. We found an IPC increment up to 11.5%; in average the increment is about 5%.

In Figure 6 (b) the benefits introduced by fast-decay are shown. Except for few benchmarks (lame, tiff2bw, pgpencode and rijndaelencode) the filter cache is powerful in reducing the additional latency introduced by decay technique. For the benchmarks with different behaviour, preliminary results with a



(a)



(b)

Figure 6. IPC across MiBench suite: higher is better (a) Comparison of IPC for fast-drowsy and drowsy: the baseline is the IPC achieved with drowsy (drowsy itself in average produces the 99% of IPC achieved without any power-saving technique); (b) Comparison of leakage saving in fast-decay and decay: the baseline is the IPC achieved with decay (decay itself in average produces the 80% of IPC achieved without any power-saving technique).

64KB show that the problems should be overcome. In general, decay caches may incur into a not negligible performance loss because an hit into a line that is gated introduces a load from the lower level of memory. With the filter cache the access pattern to the cache changes and, moreover, it captures a lot of access. In average, the IPC increases by a 4%.

The effectiveness our proposal, strictly depends on the miss-rate of L0 cache; the higher is the number of accesses it captures, the lower is the execution time. The miss-rate of the filter cache is shown in Table 3.

The miss-rate varies widely across the benchmark and it mainly depends on the behavior of the benchmark. However it is useful to underline that the L0, although small, is able to capture many accesses.

5.3 Leakage energy-delay product

We evaluate the energy-delay product, [20], in a way that is appropriate for our study [27, 28], as introduced in §2.1. Since we are interested in the leakage, we calculate the product as:

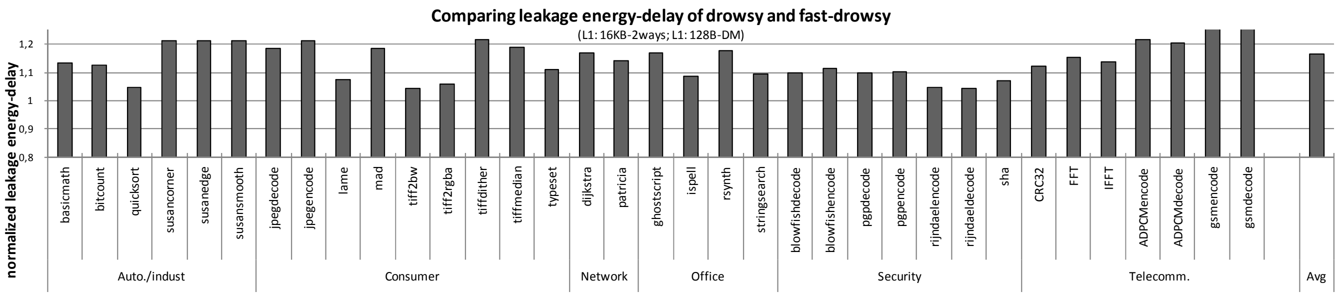
*leakage energy-delay=leakage energy * execution time*

Figure 7 shows the ratio of leakage energy-delay for fast-drowsy and fast-decay respectively compared to drowsy and decay. Figure 7 (a) shows that fast-drowsy technique reduces leakage energy-delay for all MiBench programs, while Figure 7 (b) shows the same behaviour for fast-decay except for three benchmark. This is not surprising, because these three benchmarks are the ones previously analyzed in Section 5.1 and 5.2 that don't have benefits in leakage saving and IPC.

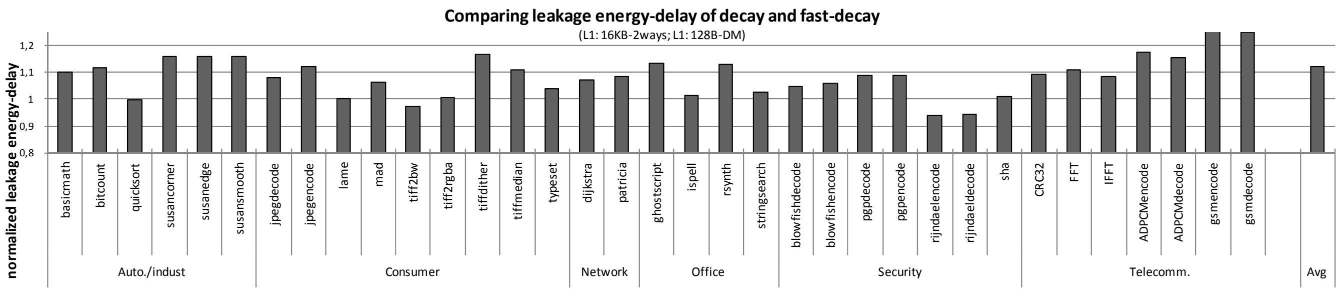
The leakage energy-delay metric emphasizes the effectiveness of the technique because it considers at the same time the two aspects we are focusing on: low-leakage and performance. In average, fast-drowsy reduces leakage energy-delay of about 17% relative to drowsy while fast-decay 13% relative to decay.

Table 3. Filter cache miss-rate

Auto./indust	basicmath	30.29%
	bitcount	23.34%
	quicksort	27.19%
	susancorner	18.93%
	susanedge	18.93%
	susansmooth	18.93%
Consumer	jpegdecode	52.27%
	jpegencode	35.77%
	lame	37.62%
	mad	34.77%
	tiff2bw	18.07%
	tiff2rgba	21.21%
	tiffdither	36.93%
	tiffmedian	21.75%
	typeset	39.04%
Network	dijkstra	55.89%
	patricia	29.06%
Office	ghostscript	34.96%
	ispell	27.52%
	rsynth	34.85%
	stringsearch	30.93%
Security	blowfishdecode	21.59%
	blowfishencode	21.59%
	pgpdecode	9.39%
	pgpencode	10.70%
	rijndaeldecode	53.91%
	rijndaelencode	50.09%
	sha	15.45%
Telecomm.	ADPCMdecode	28.68%
	ADPCMencode	31.47%
	CRC32	21.49%
	FFT	28.19%
	gsmdencode	21.90%
	gsmecode	12.24%
	IFFT	31.39%



(a)



(b)

Figure 7. IPC across MiBench suite: higher is better (a) Comparison of IPC for fast-drowsy and drowsy: the baseline is the IPC achieved with drowsy (drowsy itself in average produces the 99% of IPC achieved without any power-saving technique); (b) Comparison of leakage saving in fast-decay and decay: the baseline is the IPC achieved with decay (decay itself in average produces the 80% of IPC achieved without any power-saving technique).

6. RELATED WORK

Leakage power in cache memories is more important than dynamic with current and next generation technologies [3] [29]. New high-K materials [30] are only delaying the leakage problem, and in any case architectural solutions to cope with leakage could be usefully employed. Unfortunately the fastest implementation is not always the most beneficial from the energy stand-point [31]. Cache utilization varies widely across a range of applications and it varies significantly also during the execution of a single application [16], so there are a lot of opportunities for switching off cache lines in order to reduce leakage.

Recent studies propose a better organization of the cache by reducing its size dynamically [32]. Unused lines can be put into a low-leakage state. When a cache block is put in power-saving state, the technique is called state-preserving if the block content is maintained and, on the other hand, non state-preserving if it is destroyed [4, 15]. The main architectural methods of those two categories are drowsy caches [5] for the first one and cache decay [19] for the second one. Cache Decay uses the circuital gated-Vdd technique [6]; it introduces an extra transistor that gates the supply of the cache SRAM cells. A dramatic reduction of the leakage current is achieved, but the loss in performance is not negligible and it causes some increase in dynamic power dissipation. On the other hand, drowsy caches decrease leakage by reducing the power supply without losing information. No additional access to lower memory level is necessary during an access into a drowsy line, but the leakage reduction is smaller than in gated-Vdd. A comparison between these two proposals has been done in [15].

After these two techniques have been introduced, many others have suggested their improvements. In [13], a drowsy cache policy called “Reuse Most Recently used On (RMRO)” is proposed. It is an advanced technique to select the lines to switch off based on their usage. Drowsy cache approach is used with Region based cache in [22], allowing both leakage saving and no performance overhead. Multiple states with progressive reduction of voltage supply before the data loosing are proposed in [12]. A detailed FSM is presented in order to select the more useful supply. The application of the drowsy technique to the instruction cache has been studied in [9]. Meng et al. [18] explored the limit of leakage power reduction in caches and they found that, with the perfect knowledge of the access pattern, it is possible to find the exact moment when to put a line into drowsy state or when to switch it completely off.

A trade-off approach between performance and *dynamic* energy consumption, has been proposed in [17]. A tiny filter cache is positioned behind the processor and the standard L1 data cache to reduce the performance loss. Improvements of this technique has been proposed in [24, 33-35].

7. CONCLUSION

This paper proposes to further reduce leakage by shortening the lifetime of high power lines in a power-saving capable cache. We have used a simple and effective solution: the addition of a tiny filter “L0” cache placed between CPU and the first level of power-saving capable data cache. This solution, although simple, has never been explored for reducing leakage power, as of our knowledge, but only to save dynamic power. The objective here is to exploit the filtering action provided by filter cache for mostly used data in order to allow the leakage power-saving policy in L1 to work better. Our results, obtained across all MiBench benchmarks show that fast-drowsy (L0+L1drowsy) versus drowsy (L1drowsy) increments of about 5% the IPC, reduces of about 10% the leakage and of about 17% the leakage energy-delay (in average); fast-decay (L0+L1decay) versus decay (L1decay) provides 4% of increment in IPC and reduces of about 6% the leakage and 13% the leakage energy-delay (in average). Results for leakage energy-delay are shown in Figure 8.

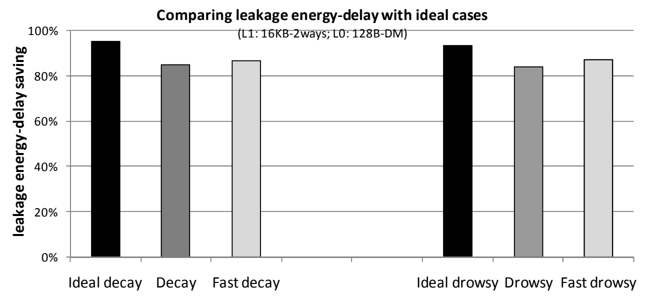


Figure 8. Comparison of leakage saving of decay/drowsy and fast decay/drowsy with theoretical ideal cases. Our proposal reduces the gap (full Mibench suite average).

7.1 Future work

In the future, we will extend this work by exploring several other parameters in the design space (e.g., policies of the power-saving technique, cache sizes, benchmarks).

8. ACKNOWLEDGEMENTS

We are particularly grateful to Prof. Sally McKee form Cornell University for providing us with a modified version of HotLeakage retargeted for ARM ISA. This research is also supported by the HiPEAC Network of Excellence.

9. REFERENCES

[1] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility (poster session)", *International Symposium on Low Power Electronics and Design (ISLPED'00)*, Rapallo, Italy, 2000, pp. 241-243.

[2] S. Segars, "Low power design techniques for microprocessors", *International Solid-State Circuits Conference Tutorial (ISSCC'01)*, 2001

[3] A. Allan, D. Edenfeld, W. H. Joyner Jr, A. B. Kahng, M. Rodgers, and Y. Zorian, "2001 technology roadmap for semiconductors", in *Computer*, vol. 35, pp. 42-53, 2002.

[4] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron, "State-preserving vs. non-state-preserving leakage control in caches", *Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, 2004, pp. 1530-1591/04.

[5] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power", *Annual International Symposium on Computer Architecture (ISCA'02)*, 2002, pp. 148-157.

[6] 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", *International Symposium on Low Power Electronics and Design (ISPLED'00)*, Rapallo, Italy, 2000, pp. 90--95.

[7] B. Allu and W. Zhang, "Static next sub-bank prediction for drowsy instruction cache", *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, 2004, pp. 124-131.

[8] M. J. Geiger, S. A. McKee, and G. S. Tyson, "Drowsy region-based caches: minimizing both dynamic and static power dissipation", *2nd Conference on Computing Frontiers (CF'05)*, Ischia, Italy, 2005, pp. 378--384.

[9] 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", in *Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'02)*, pp. 1072-4451/02, 2002.

[10] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Single-v_DD and single-v_T super-drowsy techniques for low-leakage high-performance instruction caches", *Proceedings of the 2004 international symposium on Low power electronics and design*, Newport Beach, California, USA, 2004, pp. 54--57.

[11] Y. Meng, T. Sherwood, and R. Kastner, "On the limits of leakage power reduction in caches", *International Symposium on High-Performance Computer Architecture (HPCA'05)*, 2005, pp. 154-165.

[12] N. Mohyuddin, R. Bhatti, and M. Dubois, "Controlling leakage power with the replacement policy in slumberous caches", *Conference on Computing Frontiers (CF'05)*, Ischia, Italy, 2005, pp. 161--170.

[13] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, "Exploiting temporal locality in drowsy cache policies", *Conference on Computing Frontiers (CF'05)*, Ischia, Italy, 2005, pp. 371--377.

[14] L. Li, I. Kadayif, Y. F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam, "Leakage Energy Management in Cache Hierarchies", *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques* 2002, pp. 131-140

[15] D. Parikh, Y. Zhang, K. Sankaranarayanan, K. Skadron, and M. Stan, "Comparison of State-Preserving vs. Non-State-Preserving Leakage Control in Caches", in *Workshop on Duplicating, Deconstructing and Debunking (held in conjunction with ISCA'03)*, pp. 14–25, 2003.

[16] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial Memory Streaming", *Annual International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 252-263.

[17] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure", *Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'97)*, 1997, pp. 184-193.

[18] Y. Meng, T. Sherwood, and R. Kastner, "Exploring the limits of leakage power reduction in caches", in *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 2, pp. 221-246, 2005.

[19] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power", *Proceedings of the 28th annual international symposium on Computer architecture*, 2001, pp. 240-251.

[20] H. H. S. Lee, J. B. Fryman, A. U. Diril, and Y. S. Dhillon, "The Elusive Metric for Low-Power Architecture Research", in *Workshop on Complexity-Effective Design in conjunction with ISCA-30 (WCED'03)*, 2003.

[21] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects", University of Virginia Tech report, Charlottesville 2003

[22] M. J. Geiger, S. A. McKee, and G. S. Tyson, "Beyond Basic Region Caching: Specializing Cache Structures for High Performance and Energy Conservation", *International Conference on High Performance Embedded Architectures & Compilers (HiPEAC'05)*, Barcelona, Spain, 2005, pp. 48109--2122.

[23] M. R. a. R. Guthaus, J.S. and Ernst, D. and Austin, T.M. and Mudge, T. and Brown, R.B., "MiBench: A free, commercially representative embedded benchmark suite", *Annual Workshop on Workload Characterization (WWC'01)*, 2001, pp. 83--94.

[24] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using dynamic cache management techniques to reduce energy in a high-performance processor", *International symposium on Low power electronics and design (ISLPED'99)*, 1999, pp. 64-69.

[25] K. Vivekanandarajah, T. Srikanthan, S. Bhattacharya, and P. V. Kannan, "Incorporating pattern prediction technique for energy efficient filter cache design", *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03)*, 2003, pp. 44-47.

[26] Intel, "Intel XScale Microarchitecture", in *Technical Summary*, 2000.

[27] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage indeep-submicron high-performance I-caches", in *The Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, pp. 147-157, 2001.

[28] M. Powell, S. H. Yang, B. Falsafi, K. Roy, and N. Vijaykumar, "Reducing leakage in a high-performance deep-submicron instructioncache", in *IEEE Transactions on Very Large Scale Integration Systems (VLSI'01)*, vol. 9, pp. 77-89, 2001.

[29] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power", in *Computer*, vol. 36, pp. 68-75, 2003.

[30] J. W. McPherson, "Reliability challenges for 45nm and beyond", in *Annual Conference on Design Automation (DAC'06)*, pp. 176-181, 2006.

[31] L. Benini, A. Macii, E. Macii, and M. Poncino, "Analysis of Energy Dissipation in the Memory Hierarchy of Embedded Systems: A Case Study", *Mediterranean Electrotechnical Conference (MEleCon'00)*, Lemesos, Cyprus, 2000, pp. 236-239.

[32] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation", in *Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'99)*, pp. 248–259, 1999.

[33] K. Vivekanandarajah, T. Srikanthan, and S. Bhattacharyya, "Decode Filter Cache for Energy Efficient Instruction Cache Hierarchy in Super Scalar Architectures", *Conference on Asia South Pacific design automation: electronic design and solution fair (ASDAC'04)*, 2004, pp. 373-379.

[34] M. G. Kabadi and R. Parthasarathi, "Live-Cache: Exploiting Data Redundancy to Reduce Leakage Energy in a Cache Subsystem", in *Lecture Notes in Computer Science (LNCS'03)*, pp. 337-351, 2003.

[35] C. Yang and C. H. Lee, "HotSpot Cache: Joint Temporal and Spatial Locality Exploitation for I-Cache Energy Reduction", *International Symposium on Low Power Electronics and Design (ISLPED'04)*, 2004, pp. 114-119.