Leakage Energy Reduction for Hard Real-Time Caches

Yijie Huangfu and Wei Zhang
Department of Electrical and Computer Engineering
Virginia Commonwealth University
Richmond, VA 23284
{huangfuy2,wzhang4}@vcu.edu

***Abstract*—Cache leakage reduction techniques usually compromise time predictability, which are not desirable for real-time systems. In this work, we extend the cache decay and drowsy cache techniques within the hardware-based Performance Enhancement Guaranteed Cache (PEG-C) architecture. The PEG-C can dynamically monitor the performance penalties caused by using leakage energy reduction techniques to ensure that the worst-case execution time (WCET) is better than the case without using any cache. At the same time, the leakage energy of caches can be reduced significantly.**

Keywords—Cache Memory, Leakage Reduction, Real-time Processor Architecture

I. INTRODUCTION

In modern processors, cache memories are widely used to mitigate the speed gap between the memory and the processor for improving the average-case performance. However, cache memories are harmful to time predictability, because the latency to access instructions/data depends on whether or not the access hits in the cache, which is hard to predict statically. As a result, cache memories are not attractive for real-time systems, especially for hard real-time and safety-critical systems. In those systems, time predictability, and particularly predicting the worst-case execution time (WCET), is critical for ensuring schedulability and safety.

There have been many research efforts on WCET analysis for both instruction and data caches [8], [9], [10], [11]. However, estimating the WCET of a cache memory requires complex analysis to model its dynamic behavior, which is generally hard to predict in a safe, yet not over-pessimistic way [12].

At the same time, cache memories have become major targets for energy optimization, because they typically consume a large fraction of on-chip real estate. With the scaling of technology and the decrease of supply voltage, it becomes increasingly important to reduce the cache leakage energy to improve the overall energy efficiency. Researchers have proposed leakage energy management techniques to gate or lower the power supply of the cache lines that are possibly in idle state, which are known as the cache decay [1] and the drowsy cache [2] respectively.

Although the proposed leakage management approaches [1], [2] can effectively reduce the cache leakage energy, they further complicate the timing behavior of the cache. For the cache decay, if data in a decayed cache line are needed again, they have to be fetched from the lower-level memory, which can take much longer time. For the drowsy cache, the cache lines in the drowsy mode must be activated before they are accessed, resulting in additional clock cycles. As a result, both leakage energy management techniques [1], [2] make the cache WCET analysis more complex, because the WCET analyzer now has to model all possible cache energy states for all the cache lines and all possible access sequences, making it extremely hard if not impossible to derive the WCET accurately and safely.

In order to apply the cache decay and drowsy cache techniques to reduce cache leakage energy consumption for real-time systems, we propose a pure hardware-controlled cache architecture that can provide bounded WCET. The PEG-C can leverage runtime performance monitoring based on cost/benefit analysis to ensure performance enhancement even in the worst case, including use cache leakage reduction techniques. In a PEG-C, both the cache decay and drowsy cache techniques can be safely applied to reduce the cache leakage energy, while at the same time, the worst-case execution time of PEG-C is guaranteed for hard real-time systems.

II. BACKGROUND

A. Cache Decay vs Drowsy Cache

There are two widely used mechanisms to reduce cache leakage energy, including the cache decay [1] and the drowsy cache [2]. The cache decay technique [1] manages each cache line individually and gates off the power supply to a cache line if it has not been accessed for a certain number of cycles, which is called the *decay interval* and is configurable. The power supply of a cache line keeps off until there is a new cache access that needs to use this cache line, at which point it is activated. Since the cache decay gates off the power supply, the decayed cache line does not consume any leakage energy. However, if the data in the decayed cache line is needed again, it has to be fetched from the lower-level memory, for example the L2 cache, which can take much longer latency. Therefore, an effective cache decay management needs to consider both leakage energy saving and potential performance penalty.

In the drowsy cache [2], all the cache lines are periodically turned into a low-power drowsy state with reduced power supply voltage while still maintaining the data content. Cache lines in the drowsy states consume much less leakage energy. However, drowsy cache lines must be activated before they can be accessed, which can affect performance.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

TABLE I. PARAMETERS OF ENERGY MODEL

E_l	Leakage Energy
E_{l_decay}	Leakage Energy in Cache Decay model
E_{l_drowsy}	Leakage Energy in Drowsy Cache model
P_b	Leakage Power of the cache bank
P_t	Leakage Power of the cache tag of a bank
P_d	Power of the data block of a bank
T	Execution time
E_d	Dynamic energy
E_{rw}	Energy consumption per read/write access
N_{rw}	Read/write access times
T_{act}	Time in active mode of a cache bank
T_{drowsy}	Time in drowsy mode of a cache bank
α_{decay}	Energy reduced ratio in decay model
α_{drowsy}	Energy reduced ratio in drowsy model
E_{oh_decay}	Energy overhead in decay model
E_{oh_drowsy}	Energy overhead in drowsy model
K_{decay}	Decay activation energy consumption factor
K_{drowsy}	Drowsy activation energy consumption factor
N_{act}/N_{deact}	Activation/Deactivation times
T_p	Length of the clock period
E_{al}	Energy per access to the lower level memory
N_{al}	Access to the lower level memory caused by the decay power policy

Both the cache decay and drowsy cache are shown to have small hardware and energy overheads. More details about both techniques can be found in [1] and [2] respectively.

B. Energy Model

Table I shows the meaning of the parameters that are used in the following cache decay and drowsy cache energy model. Without using any power management policy, the leakage and dynamic energy consumption of a cache memory are calculated by

$$E_l = P_b * T \tag{1}$$

$$E_d = E_{rw} * N_{rw} \tag{2}$$

Then the energy consumption of the cache is $E = E_d + E_l$.

In the cache decay model, the leakage energy consumption is calculated by

$$E_{l\_decay} = P_b * T_{act} + P_b * T_{decay} * \alpha_{decay} \tag{3}$$

The energy overhead introduced by the cache decay policy is

$$E_{oh\_decay} = K_{decay} * (N_{act} + N_{deact}) * P_b * T_p + E_{al} * N_{al} \tag{4}$$

Therefore, in the cache decay model, the energy consumption of the cache is $E = E_d + E_{l\_decay} + E_{oh\_decay}$.

In the drowsy cache model, the leakage energy consumption can be calculated by

$$E_{l\_drowsy} = P_t * T + P_d * T_{act} + P_d * T_{drowsy} * \alpha_{drowsy} \tag{5}$$

The energy overhead introduced by the drowsy cache policy is

$$E_{oh\_drowsy} = K_{drowsy} * (N_{act} + N_{deact}) * P_d * T_p \tag{6}$$

Thus in the drowsy cache model the total energy consumption is $E = E_d + E_{l\_drowsy} + E_{oh\_drowsy}$.

It should be noted that in our drowsy cache model, we assume the tag arrays are not put into the drowsy state to detect cache hit/miss quickly. Therefore, the leakage reduction for the drowsy cache is focused on the data arrays. By comparison, in the cache decay, since the data are lost, we can also turn off the tag arrays to further reduce the leakage energy without impacting the performance.

TABLE II. ENERGY MODEL PARAMETERS VALUES(90NM TECH NODE)

	Capacity	Dynamic access energy(nJ)	Leakage power of a bank(mW)
Cache Bank	4KB	0.0191163	2.88675
	8KB	0.0301051	5.62849
	16KB	0.0414208	10.5877
Data Array	4KB	0.013187	1.7827
	8KB	0.0198433	3.51069
	16KB	0.028706	6.64529
Tag Array	4KB	0.0059293	1.10405
	8KB	0.0102617	2.1178
	16KB	0.0127148	3.94245
L2	256KB	0.34421	103.428

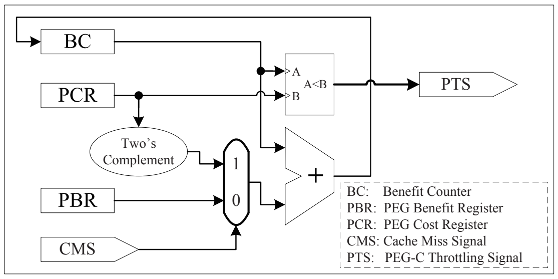


Fig. 1. Basic hardware-based PEG-C architecture.

The CACTI power model [4] is used to generate the values of static and dynamic energy and power parameters of caches with different sizes and different memory types, as shown in Table II.

III. HARDWARE-BASED PEG-C ARCHITECTURE

A. Pure PEG-C

Recently, a software-assisted PEG-C cache was introduced in [3]. This PEG-C, however, relies on software to preload initial instructions to guarantee performance improvement. Applying cache leakage reduction techniques may lead to PEG-C throttling at runtime, thus degrading the average-case performance and energy efficiency. In this paper, we propose a pure hardware-based PEG-C, which also includes an additional shadow state to avoid shutting down PEG-C completely. Moreover, we extend the PEG-C performance monitoring hardware to enable the use of cache decay and drowsy cache techniques while proving hard real-time guarantee.

Assume the system only has a main memory and a one-level cache, and the access latencies to them are L_m and L_h respectively. If there is a cache hit, the core takes L_h to get the needed data or instruction. If there is a cache miss, it takes $L_h + L_m$, because the core needs to access the cache first to detect the miss and then access the main memory. Therefore, compared with the system with only the main memory (without caches), for which the memory access

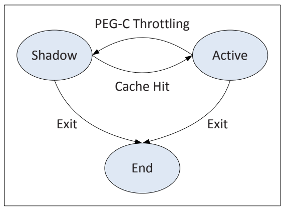


Fig. 2. The state diagram of the hardware-based PEG-C controller.

latency is always L_m, a cache miss actually degrades the performance by L_h cycles. Obviously, if there are too many cache misses, which might happen in the worst case, the WCET with the cache can potentially become even worse than that without using the cache. As a result, in many hard real-time systems, designers simply choose to disable the caches to ensure time predictability [7].

Therefore, to use a cache memory safely to benefit hard real-time systems, it is important to guarantee that the WCET is at least not worse than the baseline case without using a cache, which is called the Performance Enhancement Guaranteed (PEG) property in this paper. The proposed hardware-based PEG-C can achieve this by simple hardware extensions with the runtime performance monitoring mechanism as shown in Figure 1.

The PEG-C performance monitoring is based on efficient performance cost/benefit analysis at runtime. If there is a cache hit, $L_m - L_h$ cycles are saved as compared to the baseline case without any cache. If there is a miss, L_h cycles are wasted. Thus, the PEG-C uses a *Benefit Counter (BC)* register to monitor the performance benefit of the system, and uses a *PEG-C Benefit Register (PBR)* and a *PEG-C Cost Register (PCR)* to store the configurable values of the performance benefit and cost respectively, i.e. $L_m - L_h$ for *PBR* and L_h for *PCR*.

The PEG-C has three states, including the shadow state, the active state, and the end state, as shown in Figure 2. At the beginning of the execution of a program, the PEG-C is in the shadow state, in which the data/instruction is directly fetched from the main memory as if there is no cache. But the content of the cache is still checked and updated accordingly. The PEG-C switches from the shadow state to the active state whenever there is a cache hit. In the active state, the PEG-C operates just like a regular cache. The only difference is that the value in *BC* is updated according to the following rules.

Under a hit, *BC* is added by the value in *PBR*, whereas under a miss, *BC* is subtracted by the value in *PCR*. Whenever the value of *BC* is less than the value in *PCR* in the active state, which indicates a further miss would lead to performance worse than the baseline, the PEG-C is throttled. In this case, the PEG-C is switched back to the shadow state, and instructions/data are directly fetched from the main memory. The PEG-C is kept in the shadow state until there is a cache hit again or until the program finishes as depicted in Figure 2.

It should be noted that The hardware-based PEG-C is complementary to the software-based WCET analysis techniques [8], [9], [10], [11] that have made significant progress in the past two decades. The WCET analysis techniques can be applied to PEG-Cs to derive tighter WCET, as the PEG-Cs do not degrade performance as compared to regular caches. On the other hand, the PEG-C can be used in cases that the WCET is too complicated or not applicable. For instance, the cache leakage management techniques can significantly complicate the WCET analysis, while the PEG-C can enable them.

B. PEG-C with Leakage Energy Management

When the leakage management mechanisms are applied to the PEG-C to reduce leakage energy, some additional rules to update the *BC* are added to ensure the PEG property. Figure 3 shows the architecture of the PEG-C with the drowsy cache policy, in which the additional register and signal are shown in red color. In the drowsy PEG-C, extra cycles are needed to activate a cache line from the drowsy state. Therefore the number of clock cycles needed by activations is subtracted from the *BC* when the PEG-C is in the active state and there is a cache hit. If the cache access is a miss, the activation cycles can be overlapped with the main memory access latency, hence no further value needs to be subtracted from the *BC*.

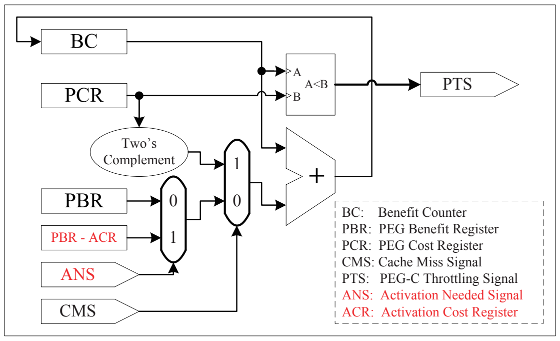


Fig. 3. Hardware-based PEG-C for drowsy cache.

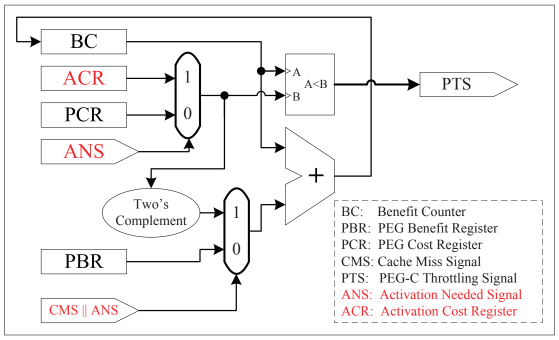


Fig. 4. Hardware-based PEG-C for cache decay.

The PEG-C extension for the cache decay is shown in Figure 4. If a cache line in the decay state is accessed, it must be activated first. However, the activation cycles can be overlapped with the main memory access latency. Therefore, an access to a cache line in the decay state can be treated as a cache miss in the PEG-C. In case there is a real cache miss, the performance penalty is the value stored in *PCR*, while the penalty is the value stored in *ACR* (i.e. 1 cycle by default) if an activation is needed.

Since the PEG-C can guarantee the baseline performance in the worst case, the baseline performance can be used as a reference WCET (though it may not be the exact WCET). This essentially eliminates the need of complex cache timing analysis, which is especially difficult for data cache accesses whose addresses (e.g. in the heap) may not be statically predicted. At the same time, the PEG-C does not prevent the use of existing cache timing analysis techniques [8], [9], [10], [11] to derive the WCET for the PEG-C

TABLE III. MEMORY MODEL CONFIGURATION IN TRIMARAN

	L1	L2
Cache Size	8KB	256KB
Block Size	8B	32B
Associativity	2	4
Access Latency	1	10

IV. EVALUATION

A. Evaluation Methodology

We use the Trimaran compiler/simulator framework [5] to build and evaluate the PEG data and instruction cache models with both cache decay and drowsy cache policies. Table III shows the default configuration of the memory system. The L1 D-Cache and I-Cache are PEG-Cs with different energy management policies, while the L2 D-Cache and I-Cache are just PEG-Cs without using any energy management policy. In the cache decay and drowsy cache model, the decay and drowsy intervals are varied from 1000 cycles to 2000, 4000, 8000, and 16000 cycles. The 12 benchmarks used in our experiments are selected from Mediabench [6].

B. PEG-C vs. Regular Caches

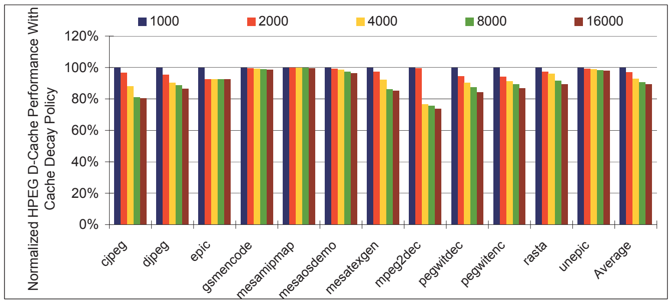
To verify the effectiveness of the PEG-C in guarding the performance when different energy policies are applied, the data and instruction cache models with both leakage energy management mechanisms are simulated. The results show that, although the cache decay and drowsy cache policies introduce performance overheads due to the activations of the cache lines from the low power states, the PEG-C can help to guarantee the baseline performance while achieving desirable leakage energy reduction.

Figures 5 and 6 contain the normalized performance results (in total execution cycles) of the PEG-Cs with the cache decay and drowsy cache management respectively, which are normalized to the total execution cycles of a regular decay or drowsy cache with a 1000-cycle drowsy or decay interval without using the PEG-C control. We can see that the PEG-C control itself does not vary the performance, because our evaluation shows that all the benchmarks only stay in the shadow mode at the beginning of the execution for a very short period of time. Thus, PEG-C can guard the cache usage to guarantee that the worst-case execution time is always better than or equal to the baseline performance.

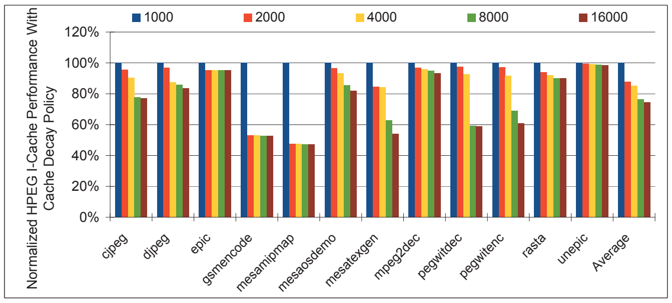
The results in Figures 5 and 6 also show that longer decay/drowsy intervals generally lead to better performance. This is because with longer intervals, fewer activations are needed. We also observe that the cache decay policy is more sensitive to the length of the intervals. This is because small interval values in the cache decay policy may lead to more lower level memory accesses, which may significantly degrade the performance.

C. Drowsy PEG-C Results

Figure 7 shows the percentages of the total execution time during which the data arrays are in the drowsy state, and Figure 8 gives the leakage reduction of the data arrays. The results clearly indicate that, with smaller drowsy intervals, the data arrays can stay in the drowsy state for longer time, which results in more leakage reduction. We also find that, on

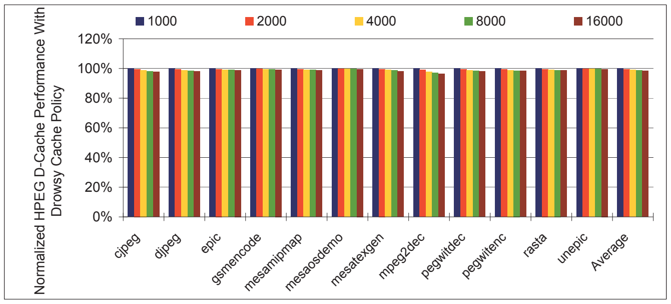


(a) Normalized performance of D-Cache

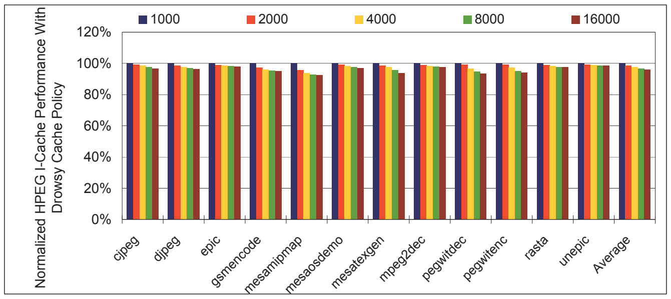


(b) Normalized performance of I-Cache

Fig. 5. Normalized performance of the PEG-C with different cache decay intervals.

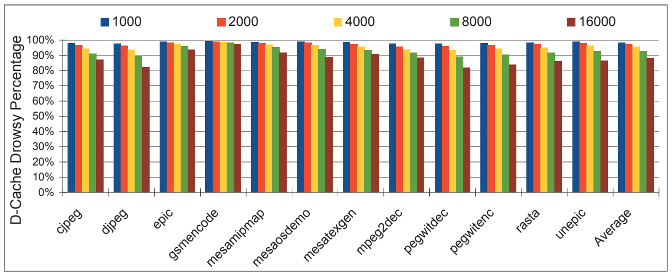


(a) Normalized performance of D-Cache

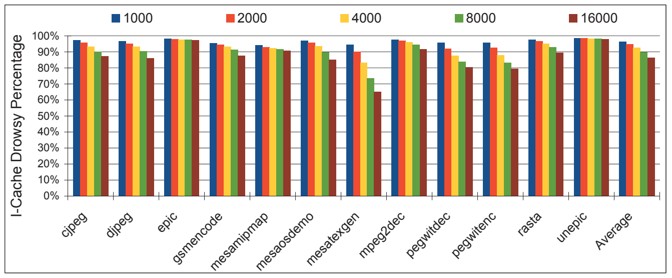


(b) Normalized performance of I-Cache

Fig. 6. Normalized performance of PEG-C with different drowsy cache intervals.

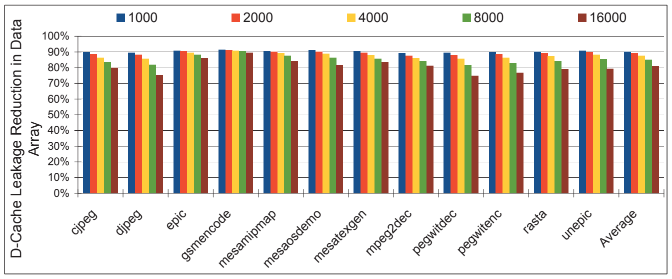


(a) Drowsy percentage of the data array in D-Cache

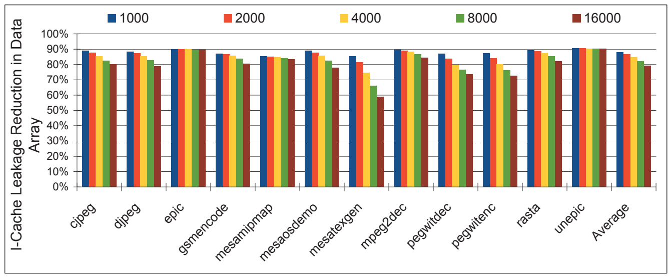


(b) Drowsy percentage of the data array in I-Cache

Fig. 7. Drowsy percentage of the data arrays.

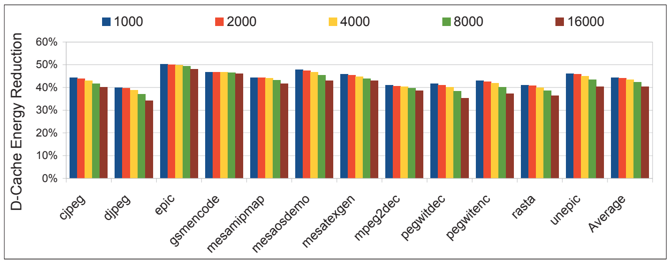


(a) Leakage reduction of the data arrays in the drowsy PEG D-Cache

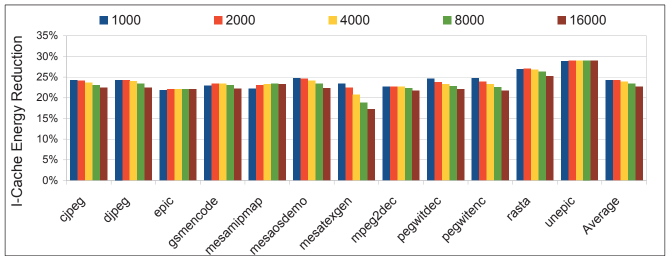


(b) Leakage reduction of the data arrays in the drowsy PEG I-Cache

Fig. 8. Leakage reduction of the drowsy data arrays.

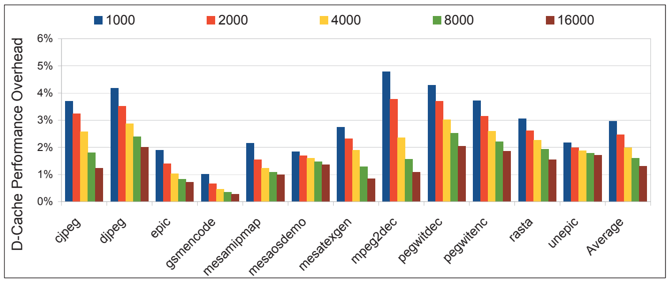


(a) Energy reduction of the drowsy PEG D-Cache

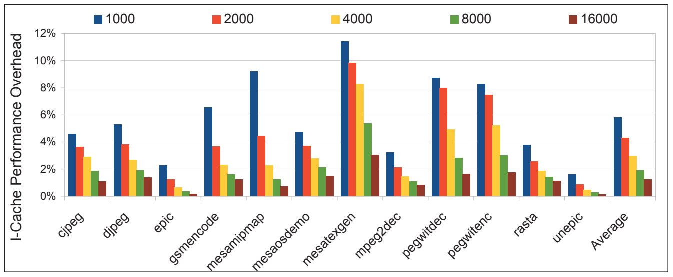


(b) Energy reduction of the drowsy PEG I-Cache

Fig. 9. Energy reduction of the drowsy PEG-C.



(a) Performance overhead of the PEG D-Cache



(b) Performance overhead of PEG I-Cache

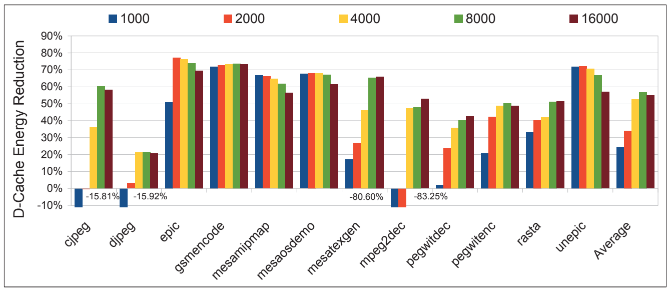
Fig. 10. Performance overhead of the drowsy PEG-C.

average, the leakage reduction for the D-Cache is better than that of the I-Cache. This is because in general the data accesses tend to have more temporal locality, leaving more data cache lines in the drowsy mode for longer time, thus leading to better leakage saving.

Figures 9 and 10 show the total cache energy reduction by considering both the data and tag arrays, and the performance overhead introduced by the drowsy policy respectively. As expected, the performance overhead decreases as the drowsy interval increases. This is because, with a longer drowsy interval, the cache is less frequently turned into the drowsy state, thus it needs less number of activations. The average results also show that the performance overheads of the drowsy I-Cache is larger than that of the drowsy D-Cache, which causes the I-Cache to have less leakage energy reduction than the D-cache.

D. Decay PEG-C Results

Figures 11 and 12 present the total energy reduction and performance overhead results respectively of the decay PEG-C. We observe hat the cache decay policy can reduce more leakage energy than drowsy cache. This is because, by gating the power supply of both the data array and tag array, the cache decay policy can reduce more leakage energy. However, the results also show that the small decay intervals may lead


(a) Energy reduction of the decay PEG D-Cache

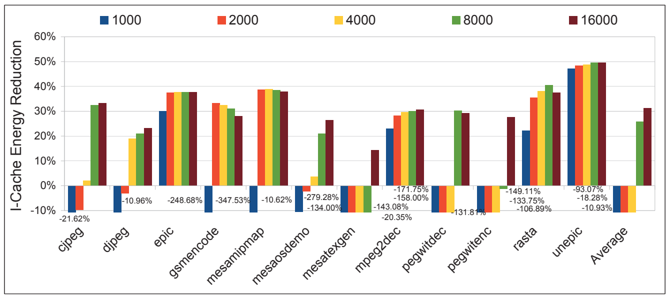
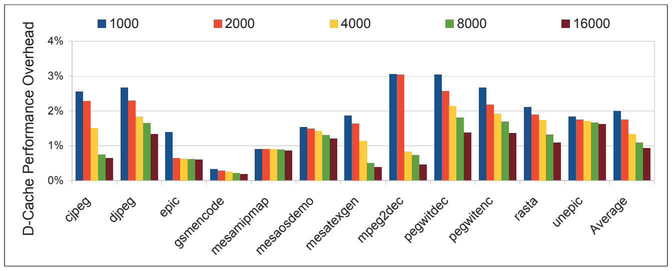

(b) Energy reduction of the decay PEG I-Cache

Fig. 11. Energy reduction of the decay PEG-C.


(a) Performance overhead of the decay PEG D-Cache

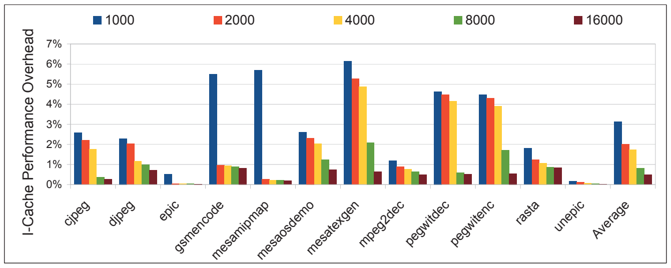

(b) Performance overhead of the decay PEG I-Cache

Fig. 12. Performance overhead of the decay PEG-C.

to very little or even negative energy reduction. This is caused by more decay cache lines and hence more frequent accesses to the lower level memory due to the smaller decay intervals.

V. Conclusion

In this work, we propose the performance enhancement guaranteed cache to enable the safe use of cache decay and drowsy cache for real-time systems. Our experimental results show that the proposed PEG-C architecture, coupled with cache leakage management techniques, can help to reduce 40% - 50% cache energy consumption while still ensuring the worst-case performance. We believe the proposed PEG-C is a useful cache design option for real-time systems, especially for hard real-time systems to improve energy efficiency without compromising time predictability.

Acknowledgment

This work was funded in part by the NSF grant CNS-1421577.

References

[1] S. Kaxiras, et al. Cache decay: exploiting generational behavior to reduce cache leakage power. In Proc. of ISCA, 2001.
[2] K. Flautner, et al. Drowsy caches: simple techniques for reducing leakage power. In Proc. of ISCA, 2002.
[3] Y. Huangfu and W. Zhang. PEG-C: Performance Enhancement Guaranteed Cache for Hard Real-Time Systems. IEEE Embedded Systems Letters (ESL), Vol. PP, Issue 99, Jan 2nd, 2014.
[4] CACTI Homepage: www.hpl.hp.com/research/cacti/
[5] Trimaran Homepage: www.trimaran.org
[6] C. Lee, et al. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In Proc. of Micro, 1997.
[7] B. Jacob. Cache Design for Embedded Real-Time Systems. In Proc. of the Embedded Systems Conference, June, 1999.
[8] R. Arnold, F. Mueller, D. Whalley and M. Harmon. Bounding Worst-case Instruction Cache Performance. In Proc. of RTSS, 1994.
[9] M. Alt, C. Ferdinand, F. Martin and R. Wilhelm. Cache behavior prediction by abstract interpretation. In Proc. of SAS, 1996.
[10] B. Lesage, D. Hardy and I. Puaut. WCET analysis of multi-level set-associative data caches. In WCET Workshop, 2009.
[11] B. Huynh, L, Ju and A. Roychoudhury. Scope-Aware Data Cache Analysis for WCET Estimation. In Proc. of RTAS, 2011.
[12] L. Wehmeyer and P. Marwedel. Influence of Memory Hierarchies on Predictability for Time Constrained Embedded Software. In Proc. of DATE, 2005.
[13] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In Proc. of RTSS, 1999.