Profile-Based Adaptation for Cache Decay: A Technical Report
UVA CS TR CS-2004-14
April 2004

Karthik Sankaranarayanan and Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
E-mail: {karthick,skadron}@cs.virginia.edu

Abstract

'Cache decay' is a set of leakage-reduction mechanisms that put cache lines that have not been accessed for a specific duration into a low-leakage standby mode. This duration is called the decay interval, and its optimal value varies across applications. This paper provides an extended discussion of the results previously presented in our journal paper [13]. It describes an adaptation technique that analytically finds the optimal decay interval through profiling, and shows that the most important variables required for finding the optimal decay interval can be estimated with a reasonable degree of accuracy using profiling. This work explicitly trades off the leakage power saved in putting both *the 'live' and 'dead' lines into standby mode, against its performance and energy costs. It achieves energy savings close to what can be obtained with an omniscient choice of per-benchmark optimal decay interval.*

1 Introduction

Energy-efficiency has become a first-class design constraint as a way to improve battery life in mobile and embedded devices and as a way to reduce operating costs in desktop and server systems. Active or 'switching' power has been the dominant part of processor power. However, as process technology and threshold voltages scale, static or 'leakage' power has grown in importance, growing exponentially [18] and possibly approaching 50% of total power dissipation within the next two or three technology generations. Since L1 caches form a significant portion of processor power (about 15-20%), there has been a focus on techniques that reduce their leakage power by shutting down idle cache lines.

Cache decay [6] is one such technique that shuts down infrequently accessed cache lines to save leakage power. When a cache line has not been accessed for some time—the *decay interval*, it is considered dead and put in a standby state. The decay interval is important for non-state-preserving techniques like *gated-V_{dd}* [6], where putting a line into standby consists of shutting off power to that cell, and an *induced miss* incurs the cost of an extra access to the next level of cache (this has also been referred to as gated-V_{ss}). The decay interval is much less important for state-preserving techniques like drowsy cache [4], but recent work has suggested that L1 cache decay using gated-V_{dd} is more energy-efficient and has less performance cost for the latencies seen with on-chip L2 caches [11]. The energy and performance advantage of gated-V_{dd} is likely to be even better than that work suggested, because it did not account for the energy and performance cost of replay traps that drowsy-cache 'slow hits' would incur, a problem that is not present for gated-V_{dd}.

With gated-V_{dd}, too-short decay intervals lead to many live lines being shut down erroneously. Too-long decay intervals lead to dead or long-unused lines not being shut down and hence resulting in little or no leakage power savings. Hence, a correct choice of the decay interval is important. The optimal value of this decay interval varies across applications and across program phases within an application. Decay-based leakage saving techniques should *adapt* the decay interval to the value optimal to the particular application's behaviour. This paper uses profiling and off-line analysis to determine the optimal value of the decay interval on a per-application granularity for maximizing energy savings.

Related work Kaxiras et al.'s work on cache decay [6] improves upon a coarser-granularity work [21] on instruction caches, which shuts down groups of I-cache lines. Both these techniques use the gated-V_{dd} technique [12], which uses an extra 'sleep transistor' in a standard SRAM cell. The sleep transistor gates off the power supply to the cell on a 'sleep signal' thereby reducing the leakage current to almost zero. Since the power supply of the SRAM cell is cut off, gated-V_{dd} is a non-state-preserving technique and hence, on powering the cache line back up, data has to be read from a lower level cache or memory. Other state-preserving alternatives to gated-V_{dd} like reverse body bias [7, 10] and drowsy caches [4] have been proposed, but may be inferior for L1 caches that are backed by fast, on-chip L2 caches [11]. Our work is mainly based on cache decay that uses gated-V_{dd}. Although the benefit is greatest for non-state-preserving techniques, time-based decay using any of these techniques involves a decay interval and hence our method can be applied to all the above techniques.

Adaptation of decay interval according to the application behaviour has been looked at by Kaxiras et al. [6], Zhou et al. [23] and Velusamy et al. [20]. Kaxiras et al. use a per-line adaptation scheme that adapts the decay interval of each line according to the time at which a miss occurs after shutting down a line. Zhou et al.'s Adaptive Mode Control (AMC) and Velusamy et al.'s Integral Miss Controller (IMC) are global schemes which adapt the decay interval at the granularity of the whole cache. They keep the tag array always powered on to measure the application's *true* behaviour without decay. AMC adapts the decay interval by making the application's decay behaviour track its *true* non-decay behaviour while IMC uses formal feedback control to enforce a limit on the performance degradation. While all these techniques indirectly exploit shutting down both live and dead lines to save leakage power, the performance versus energy trade-off is not made explicit by them. For IMC and AMC, it lies hidden under the tunable parameters of the schemes (set-point for IMC and Performance Factor for AMC). For per-line adaptive decay, it depends on the choice of its decay interval. Further, IMC and AMC also suffer from the difficulty in tuning these parameters according to application behaviour. In this paper, we compare our work to each of the above adaptation techniques.

Contributions This paper provides a profile-based technique for decay interval adaptation that chooses the optimal decay interval through offline analysis. We find that the important variables that determine the energy saved due to decay can be estimated with a reasonable degree of accuracy through profiling. Further, this work explicitly considers the energy versus delay trade-off of shutting down live lines. It performs almost as well as an 'energy optimal' omniscient choice of per-benchmark individual-best decay interval. A similar framework should be easy to apply to other cache-like structures like the L1 I-cache, L2 cache, branch-prediction structures, trace cache, value predictor, etc.

This paper is an extension of our journal paper [13] and allows us to present a more elaborate discussion of the results of our work. The remainder of the paper is organized as follows: Section 2 details the background and motivation for our work, Section 3 describes our estimation technique, Section 4 describes the evaluation framework we have used, Section 5 discusses the results and Section 6 concludes the paper.

2 Background and motivation

2.1 Cache decay

Once a new line is brought into a cache, it typically gets accessed many times due to the locality of the program. After the last access, it remains in the cache until a new line replaces it. During the time after its last access and before eviction, it idles in the cache, dissipating leakage power. The time between the arrival of the cache line or the beginning of a new *generation* and the last access to it is called the *live time* of the generation. The time between two successive accesses to the same line is called an *access interval* of the generation. The idle period before eviction is called the *dead time* of the generation. Cache decay [6] identifies these dead times and switches the lines into standby mode by gating off their power supply—hence saving the leakage power that would have been dissipated during those dead times.

Cache decay typically uses counters to measure the last time a cache line was accessed. If the duration since the last access is greater than a particular threshold, the line is assumed to be dead and is put into standby. This threshold used to distinguish between live versus dead lines is called the *decay interval*. If the decay interval is too small, many live lines will be put into standby. Since gated-V_{dd} is a non-state-preserving technique, shutting down live lines leads to extra, *induced* misses to the next level cache, which slow the program down. Moreover, these extra misses lead to additional dynamic energy dissipation in the processor through extra execution time and also the energy cost of re-fetching the lines. On the other hand, if the decay interval is too large, many dead lines will not be put into standby, hence dissipating unnecessary leakage power. Kaxiras et al. found that the access intervals in programs were much shorter than the dead times, thereby providing a convenient

demarcation point for the decay interval. They used a 'competitive algorithms' analysis to obtain a universal decay interval value of 8K cycles.

Leakage energy savings in cache decay is due to putting cache lines into the low-leakage standby mode. A metric that numerically measures the degree to which lines are shut down is the *turn-off ratio*. It is the fraction of time that the cache lines are in the standby mode. Lower decay intervals lead to higher turn-off ratios and vice versa. It is to be noted that higher turn-off ratios do not always translate into higher energy savings. The high turn-off ratio could come at a cost of increased performance loss or increased energy disspation due to extra misses to the lower level.

2.2 Motivation

Table 1. Access interval, dead time, ideal turn-off ratio and best decay interval statistics for a subset of SPEC2000 benchmarks.

	Access Interval (cycles)			Dead Time (cycles)			Ideal TOR (%)	Best Intvl (cycles)
	avg	stdev	max	avg	stdev	max		
art	13	27	4K	4K	6K	42K	99.02	256
twolf	254	2K	298K	29K	23K	447K	87.21	4K
eon	751	100K	363M	66K	1.8M	462M	10.93	16K

Variability The optimal decay interval for an application depends on the nature of its memory accesses. As application behaviour varies, the optimal decay interval also varies. Table 1 shows a sample of this variability for three SPEC2000 benchmarks. Two of these three benchmarks (art and eon) form the extreme cases in our study and the third (twolf) is one of the applications with 'close to average' behaviour. The first column of the table shows statistics about access intervals. The second column shows statistics about dead times. Variability *across* applications is evident in both cases. Further, not only are the means different, so are the standard deviations—i.e., the variability *within* applications is also different. The last column shows the optimal decay interval that we find in terms of energy savings. Since the applications vary so much in their memory behaviour, the optimal decay interval also varies. This suggests that there is room for adaptation in cache decay.

Live lines The penultimate column of Table 1 also shows something more interesting. It shows the dead time as a fraction of total time in these applications. Since this is the highest turn-off ratio (*tor*) that can be obtained by putting only the dead lines into standby, it is called 'ideal *tor*'. This value is very high for 'art' and 'twolf' indicating that in those applications, most of the benefit in leakage savings can be reaped by turning off dead lines. However, in 'eon', the case is different. Dead times contribute only to 11% of the total time. We also observe that, of the remaining 88% live time, more than 15% comes from access intervals that are greater than 1M cycles. However, these access intervals only form an infinitesimal (0.008%) portion of the total number of accesses. This means that, the *time contribution* of these live lines is very high (because they are long) in spite of their occurring very *infrequently*. Turning off such live lines benefits us in two ways: first, it saves a large amount of leakage energy as the lines stay off for the entire long duration of the access interval. Second, it leads to very little performance loss because of its infrequent occurrence. This indicates that, by explicitly considering the trade-off of shutting down live lines against the possible performance loss, we might potentially be able to obtain energy savings higher than otherwise.

The above two observations suggest that there is potential for an adaptive technique that explicitly considers the trade-off of putting live lines into standby mode. In order to consider such a trade-off, we take the approach of analytically deriving the variables that determine the energy savings from the access and dead time data obtained through profiling. Once these variables have been estimated, the choice of the best decay interval is simple. It is that decay interval corresponding to the variables which result in the maximum energy savings. We find that such a profile-based scheme performs almost equal to an omniscient selection of the individual-best decay interval.

3 Methodology

This section describes in detail the profile-based estimation technique used to choose the optimal decay interval. The main idea is to make use of the profile data collected to estimate the energy savings for different decay intervals and choose the interval corresponding to the maximum savings.

3.1 Normalized leakage savings

The energy savings due to cache decay are dependent on four main variables:

1. The turn-off ratio (*tor*), which is the fraction of time spent by the cache lines in the low-leakage standby mode. This is the measure of the fraction of cache leakage power saved.

2. The dynamic energy cost incurred in going to a farther cache due to turning off live lines. This consists of two parts:

   (a) When a live line is turned off, the next access to it results in a miss. This is called an *induced miss* (*im*) since it is not present in the default program behaviour without decay, but has been *induced* by the decay mechanism. Each *im* results in an access to the lower level cache which results in dynamic energy dissipation.

   (b) When a live line is turned off, it could have been dirty and could have resulted in a write-back. If this write-back is an *extra* write-back, which was not present in the default program behaviour without decay, it is called an *induced write-back* (*iwb*). It is to be noted that write-backs that occur during the decay of a live line need not always be induced. They could also be eager write-backs [8] that are just happening ahead of time. Each *iwb* also results in dynamic energy dissipation due to an access to the lower level cache.

3. The energy overhead due to any additional hardware used to implement the decay policies. In case of constant cache decay [6], this overhead is due to the counters used to measure the time of last access. In case of the adaptive decay policies IMC [20] and AMC [23], this additionally includes the energy of keeping the cache tags on since they measure the *im* by keeping the tags on.

4. The energy cost of performance degradation. In case of performance loss, since the program runs longer, additional energy is spent during these extra cycles the program takes to complete. Since the same computation is performed in the program with or without decay, this energy could either be *idling* energy (where the processor does not perform any useful computation) or *recomputation* energy (where the processor repeats some of the original computation due to reduced IPC because of pipeline stalls). Energy spent in the clock network and leakage energy in the whole processor constitute this idling energy, while energy spent in the bypass network, issue logic, queues etc. due to pipeline stalls constitutes the recomputation energy.

Among these four variables, the energy due to overhead hardware for cache decay has been shown to be negligible [6] and hence can be omitted. However, it cannot be omitted for AMC or IMC since the tag array dissipates a non-negligible fraction of the total cache power. Further, assuming ideal clock gating and neglibible recomputation energy, the energy cost of performance degradation is primarily idling energy due to leakage in the rest of the processor.

In order to express the energy savings in a normalized fashion, we take the same approach as Kaxiras et al. We express the energy savings due to decay as a ratio to baseline cache leakage energy. This leads to the following expression for the normalized cache leakage savings (*esav*)

$$esav = tor - l2\_ratio * (im\_cycle + iwb\_cycle) - idle\_ratio * perc\_slowdown \quad (1)$$

In the above expression, $l2\_ratio$ is the ratio of the dynamic energy of an L2 access per cycle to the L1 D-cache leakage energy per cycle. We use a value of 10 for this, which is the same as in [6]. im_cycle and iwb_cycle are the number of induced misses and write-backs divided by the baseline program running time in cycles. $perc\_slowdown$ is the ratio of the difference in running times with and without decay to the baseline running time. $idle\_ratio$ is the ratio of the idling energy spent in the whole of the processor per cycle to the cache leakage energy per cycle. In other words, equation 1 just says that the net energy savings due to cache decay is the difference between the leakage energy saved due to putting lines into standby mode

and the dynamic energy cost of the decay process (which mainly occurs in the form of extra accesses to the lower level cache and the extra execution time of the program).

In order to determine the value of *idle_ratio*, we use the Wattch 1.02 [2] power simulator to first find the ratio of the L1 D-cache power to the total CPU power for a cache configuration that closely resembles Alpha 21364 [19]. We use Wattch's linear scaling model for technology parameters and obtain 0.13μ numbers for V_{dd}=1.3V at a clock speed of 3 GHz. This ratio turns out to be 10.6%. We assume this ratio holds good also for leakage power. Further, assuming the rest of the processor is not using any decay policy, and using the average leakage savings value of 75% from [6], *idle_ratio* can be found to be $(100 - 10.6 * 0.75)/10.6 = 8.7$. Including the effect of non-ideal clock gating and recomputation energy spent in pipeline stalls will slightly increase this value. So, we use a value of 10 for *idle_ratio*. It is to be noted that these constants, while affecting the energy savings number, do not affect the estimation method itself.

3.2 Estimation of variables

Equation 1 shows the relationship of normalized leakage energy savings to the four variables viz. *tor*, *im_cycle*, *iwb_cycle* and *perc_slowdown*. Our goal is to obtain expressions for each of these variables as a function of the decay interval. Let us suppose that the frequency distributions of the access intervals and the dead times are available for all the decay intervals $t_1 \cdots t_n$ that we are interested in. This is not an unreasonable assumption because we could obtain it through profiling for the discrete set of decay intervals we are interested in. Note that like Kaxiras et al., we assume decay intervals to be powers of two. Hence, t_{i+1} would be double the value of t_i.

Let us begin with *im_cycle*. Induced misses are those accesses to a particular live cache line whose inter-occurrence time is greater than the decay interval. Then and only then, cache decay would have wrongly identified those live lines as dead lines. If $a(t_i)$ denotes the number of access intervals that lie between t_i and t_{i+1} (i.e., the function a is the *frequency histogram* of the access intervals), then the total number of access intervals that are greater than or equal to t_i is given by $a(t_i) + a(t_{i+1}) + \cdots + a(t_n)$. Let this be denoted by $A(t_i)$. However, if t_i is the decay interval we are interested in, then from our description of induced misses above, it is clear that the total number of access intervals greater than t_i (i.e., $A(t_i)$) is actually the number of induced misses corresponding to that decay interval t_i. Hence,

$$im(t_i) = A(t_i) = \sum_{j=i}^{n} a(t_j) \tag{2}$$

Now, let us obtain an expression for the number of decayed lines corresponding to a specific decay interval. The lines put into standby in cache decay can either be live lines or dead lines. Live lines lead to induced misses and leakage savings while dead lines lead only to leakage energy savings. If $d(t_i)$ denotes the number of dead times that lie between t_i and t_{i+1}, similar to the number of induced misses, the number of dead lines that are closed down is given by $\sum_{j=i}^{n} d(t_j)$. Let this be called $D(t_i)$. Hence, the number of decayed lines is $A(t_i) + D(t_i)$. Of those decayed lines, induced misses are the *erroneous predictions* of dead lines. Thus, the *mis-prediction rate* of the decay mechanism in identifying the dead lines is given by

$$\frac{A(t_i)}{A(t_i) + D(t_i)}$$

In order to find an expression for *iwb*, let us consider the number of access intervals that lie between t_i and t_{i+1} such that the cache line corresponding to that interval is *dirty* at the beginning of the interval. We shall denote it by $a'(t_i)$. Similarly, let $d'(t_i)$ be the number of dead times which begin with a dirty line. Then, if

$$A'(t_i) = \sum_{j=i}^{n} a'(t_j)$$

and if

$$D'(t_i) = \sum_{j=i}^{n} d'(t_j)$$

It can be seen that the number of *dirty* lines that were put into standby mode is given by $A'(t_i) + D'(t_i)$. Out of these, induced write-backs are the *mis-predictions*. Hence, with a reasonable degree of approximation, *iwb* can be obtained by multiplying this by the mis-prediction rate. i.e.,

$$iwb(t_i) = \frac{A(t_i)}{A(t_i)+D(t_i)}(A'(t_i)+D'(t_i)) \quad (3)$$

This is approximate because equation 3 does not account for eager write-backs. We will show later that in spite of such approximations, our estimation method is able to obtain the values of the variables with a reasonable degree of accuracy.

From equations 2 and 3, two of the four variables of the *esav* equation (equation 1)—*im_cycle* and *iwb_cycle* can be estimated (by dividing *im* and *iwb* respectively by the baseline execution time in cycles). The next important variable to be estimated is the turn-off ratio tor. It is determined by the time spent both by live and dead cache lines in the standby mode. To illustrate the analysis better, let us consider an example. Let the decay interval be 4K cycles. Let the number of dead times between 4K and 8K cycles be 1000 while the number of access intervals in that range be 10. Then, assuming a uniform distribution of the dead times and access intervals between 4K and 8K cycles, the expected value of the time scale—the *mid-point*, lies at the geometric mean of the end points of the range (4K and 8K cycles). We take the geometric mean here instead of the arithmetic mean because of our exponential scale of decay intervals which are successive powers of two. The value of this geometric mean is $4K\sqrt{2}$. Hence, the average *time contribution* of the dead times under consideration is 1000 x $4K\sqrt{2}$. Similarly, the time contribution of the live times is 10 x $4K\sqrt{2}$.

Let us generalize the above example. With the descriptions of $a(t_i)$ and $d(t_i)$ as frequency histograms and t_i as the decay interval, the live time contribution of the access intervals between t_i and t_{i+1} is $a(t_i)$ x t_i x $\sqrt{2}$. The total live time contribution of all access intervals greater than or equal to t_i is $\sum_{j=i}^{n} t_j a(t_j)\sqrt{2}$. Since cache decay closes down only those lines which have not been accessed for the duration of the decay interval, the standby time contribution due to live lines includes only the access intervals greater than the decay interval. Hence, if we denote $\sum_{j=i}^{n} t_j a(t_j)$ by $AT(t_i)$, then, the total standby time contribution due to switching live lines into standby mode is given by $AT(t_i)\sqrt{2}$. Similarly, if we denote $\sum_{j=i}^{n} t_j d(t_j)$ by $DT(t_i)$, then the total standby time contribution due to switching off dead lines is given by $DT(t_i)\sqrt{2}$. However, we have not considered the cost of waiting for a period equal to the decay interval before switching the lines into standby mode. This total waiting time is given by the product of the decay interval and the number of lines decayed i.e., $t_i(A(t_i)+D(t_i))$. Putting all this together, the total standby time for a decay interval t_i is given by

$$(AT(t_i)+DT(t_i))\sqrt{2} - t_i(A(t_i)+D(t_i))$$

This standby time is for all lines in the cache. Hence, the standby time per cache line is obtained by dividing this value by the number of lines in the cache, *n_lines*. The ratio of this standby time per line to the total running time of the program in cycles (*n_cycles*) gives the turn-off ratio. i.e.,

$$tor(t_i) = \frac{(AT(t_i)+DT(t_i))\sqrt{2} - t_i(A(t_i)+D(t_i))}{n\_lines * n\_cycles} \quad (4)$$

The above equation is accurate under the assumption that the access intervals and dead times are uniformly distributed in the interval t_i to t_{i+1}. Otherwise, the mean of the distribution will not lie at the geometric mean of the extremities of the range. In order to deal with this skewness in the distribution within an interval, we take a heuristic approach. We assume that the distribution is identical across all ranges. Since the distribution is identical across all ranges, its mean will lie at a constant distance from the geometric mean of the range. Hence, this mean of the distribution can be obtained by multiplying the geometric mean of the extremities by a constant multiplicative factor. The factor is multiplicative and not additive because of the exponential scale of our time range. In order to find the value of this multiplicative factor, we make use of the fact that $tor(0) = 1$ (if the decay interval is zero, then the cache is never powered on). We substitute $t_i = 0$ in equation 4. If the distribution is uniform, the right side of the equation should be equal to 1. Otherwise, it will be off by a constant factor. This factor is the corrective term that is used to multiply the $\sqrt{2}$ term in equation 4. In our estimation method, we use equation 4 with the correction described above, to deal with the skewed distribution within an interval.

From the equations above (especially 4 for tor), it can be observed that there is an *explicit* consideration of closing down of live lines. So, the above estimation could for instance, result in an explicit quantitative choice that, for a particular application, in spite of some performance loss, it is beneficial in terms of energy optimization to shut down live lines. Hence, this makes the energy versus performance trade-off more explicit and visible.

The fourth and final variable in the *esav* equation is the perfomance degradation. This is because of the induced misses and hence depends on *im_cycle*. Actually, it is the product of *im_cycle* and the average latency to fetch from the lower level. This *effective latency* is much less than the nominal access latency to the lower level: most of the access latency is hidden by the instruction-level parallelism available in the program. Further, out-of-order superscalar issue, speculation and non-blocking

caches make this effective latency variable. The effective latency also depends on the criticality of the loads which access the live lines that are turned off. Since there are many complex factors involved, we have so far been unable to come up with an analytical expression for the effective latency as a function of the decay interval. In order to make a reasonable estimate of this variable, we observed the performance degradation in going from a perfect L1 data cache to a normal data cache for all the SPEC2000 benchmarks. The extra cycles were divided by the number of L1 data cache misses to obtain the various effective latencies. Averaging them across the integer benchmarks gave a value of about 2 cycles. The floating point average was about 0.2 cycles. We use these constants as the estimates for the effective latency. This is only one of the two factors determining the performance degradation; we determine the other factor (*im_cycle*) analytically. Also, in the estimation of energy savings, merely a trend that is indicative of the best decay interval is sufficient. The absolute energy savings are not necessary to pick the right decay interval. For these reasons, we expect this treatment of effective latency not to produce dramatically different results. In fact, our results confirm this: the estimation-based prediction of the optimal decay interval matched the actual optimum with a reasonable degree of accuracy. Hence, the equation we use for performance degradation is

$$perc\_slowdown(t_i) = eff\_lat * im\_cycle(t_i) \tag{5}$$

Apart from the performance degradation, we expect some difference between the estimated and the actual values for the other variables too. This is due to the following reasons:

1. We do not directly account for eager write-backs in the analytical expressions.
2. Since the decay process slows down the program, a few access intervals and dead times may become longer than they actually are in the original program.
3. The decay process also changes the default replacement behaviour of the cache since lines that are in the standby mode get evicted before the lines that are active. For instance, if the replacement policy is LRU and if the line in the standby mode is not actually the least recently used one (this could happen with short decay intervals), clearly there is a change in the default replacement behaviour.

Since the events associated with these effects should be infrequent, we expect these differences to be negligible. Our results also confirm this expectation.

3.3 Choice of the best decay interval

From equations 1, 2, 3, 4 and 5, it can be seen that all these variables are fundamentally derived from the general access interval and dead time histograms ($a(t_i)$, $d(t_i)$) and from the histograms for dirty lines ($a'(t_i)$, $d'(t_i)$). All four of these histograms for the t_i's we are interested in (256 cycles to 1M cycles) can be generated by profiling. The profile data thus generated is analyzed as explained by the equations above to produce the energy savings (*esav*). The decay interval corresponding to the best *esav* value is chosen as the optimal decay interval. We call this the *esav-pred* method.

An alternative to choosing the best *esav* point is to go for the best ED^2 point [24]. Since DVS combined with frequency scaling can be applied to most scenarios, any power saving technique should at least be as energy-efficient as DVS, if it is to be useful. Since energy varies quadratically with voltage and to a first order approximation, delay varies linearly, ED^2 product does not change by applying DVS. If we identify the minimal ED^2 point for a power saving technique that does not use DVS, DVS can be applied over and above the technique to achieve the energy target. That way, the energy target could have been achieved at the minimal ED^2 point—or in other words, with minimal performance loss.

Since we have estimates for both energy and delay, we also attempt to find such an optimal ED^2 point in the same way we did for *esav*. However, for ED^2 product, the energy has to be that of the whole CPU and hence cannot be normalized cache leakage savings. To obtain chip-wide energy savings from normalized cache leakage savings, we use a cache leakage to CPU power ratio (*leak_ratio*) of 3%. For obtaining this number, we use the cache power to CPU power ratio of 10.6% as before and assume leakage power to be 30% of cache power. With leakage power increasing exponentially [18], this is reasonable to assume for technologies beyond 130 nm. We call this method the *ed2-pred* method.

Since delay is an important variable in the ED^2 product and since our estimate for delay is heuristic and not based on analysis, we expect that this *ed2-pred* technique may not perform well. We find that in terms of estimation accuracy, it does worse than the *esav-pred* technique.

3.4 Profiling and dynamic adaptation

In this work, we use software profiling and offline analysis. However, we believe profiling in hardware and extension to online adaptation is feasible—an interesting area for future work. In order to do the profiling in hardware and obtain the four histograms, extra table space within the CPU is required. Since decay intervals of lesser than 256 cycles and greater than 128K cycles are hardly needed, the total number of different decay intervals of interest is 10. If each table entry is a 32-bit counter, a total of 40 such counters and hence 1.3 Kbits of extra table space would be required. This is just to record the data and is comparable in energy cost to keeping the tags on as in IMC or AMC. Measurement of intervals can be done using the same counter architecture used by Kaxiras et al. The extra hardware cost could still be cost-effective because our technique chooses the correct decay interval.

Even if the extra profiling hardware is not justified, our technique could be used in a dynamic optimization framework like Dynamo [1] to collect profile data in software and then run natively in hardware. Phase detection and adaptation techniques like [5, 16] could be used to select different profiling points and the dynamic translation infrastructure (eg. [14]) could be used to run the profiler in software. Once the profiling phase is over, the optimal decay interval for the phase could be cached and re-used whenever the phase recurs. Everytime a new phase is discovered, the profiling step could be run in software for that particular phase.

4 Evaluation

This section describes the evaluation framework we use to evaluate our technique. It also explains the adaptive methods against which we compare our technique.

4.1 Simulation setup

The profiling described in the previous section is implemented in the SimpleScalar 3.0c simulator toolset [3]. The estimation equations of the previous section are implemented as perl scripts that operate on the profile data produced by the above mentioned simulator. In order to evaluate other adaptive techniques like per-line adaptive cache decay, IMC and AMC, we extend the simulator used by Velusamy et al. in their IMC work, which in turn was extended from the simulator used by Kaxiras et al. in their cache decay work. In a separate exercise, we also implemented cache decay and verified if we got results similar to Kaxiras et al.

Table 2. Configuration of the simulated processor microarchitecture.

Processor Core	
Instruction Window	80-RUU, 64-LSQ
Issue width	6 instructions per cycle (4 Int, 2 FP)
Functional Units	4 IntALU,1 IntMult/Div, 2 FPALU,1 FPMult/Div, 2 mem ports
Memory Hierarchy	
L1 D-cache size	64 KB, 2-way LRU, 64 B blocks
L1 I-cache size	64 KB, 2-way LRU, 64 B blocks both 2-cycle latency
L2	Unified, 4 MB, 8-way LRU, 128 B blocks, 12-cycle latency, WB
Memory	225 cycles
Branch Predictor	
Branch predictor	Hybrid: 4K bimod and 4K/12-bit/GAg 4K bimod-style chooser
Branch target buffer	2 K-entry, 2-way

The microarchitectural configuration we use to evaluate the techniques resembles Alpha 21364 as closely as possible. Table 2 details the microarchitectural configuration used. Finally, as described in the previous section, we use Wattch 1.02 power simulator [2] to make educated assumptions about the constants in equation 1.

4.2 Benchmarks

We evaluate our results using all the benchmarks of the SPEC2000 benchmark suite. Summary statistics are given in Table 3. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC peak settings and include all linked libraries but no operating-system or multiprogrammed behaviour. For each program, we use the train input set for profiling and the reference input set to evaluate the effectiveness of the various techniques. For the profiling runs using the train input set, we execute the programs to completion. For the evaluation runs using the reference input set, we fast-forward to a single representative sample of 1 billion instructions. The location of this sample is chosen using the data provided by SimPoint [15]. Simulation is conducted using SimpleScalar's EIO traces to ensure reproducible results for each benchmark across multiple simulations.

Table 3. Summary statistics of the benchmarks used to evaluate the techniques in this study. All benchmarks use reference inputs.

benchmark	L1 D-cache miss rate (%)	IPC	best-t (K Cycles)
INT			
mcf	26.5	0.31	1
gap	0.3	1.74	2
vpr	5.0	1.06	4
vortex	1.0	2.22	4
bzip2	1.1	2.34	4
twolf	5.9	1.52	4
gzip	1.8	2.06	8
perlbmk	0.6	2.25	8
gcc	1.0	1.97	16
crafty	0.9	2.19	16
parser	2.0	1.65	16
eon	0.1	2.02	16
FP			
art	32.8	2.30	0.25
swim	8.6	0.65	1
lucas	9.8	0.66	1
wupwise	1.7	1.53	2
applu	6.0	1.11	2
equake	10.7	0.43	2
facerec	2.6	2.25	2
mesa	0.3	2.42	4
galgel	3.0	2.57	4
ammp	4.7	1.52	4
fma3d	3.3	1.08	4
mgrid	3.4	1.24	8
sixtrack	0.2	2.46	16
apsi	2.1	1.84	16

4.3 Other adaptive techniques

The three adaptive techniques against which we compare our technique are Kaxiras et al.'s per-line adaptive cache decay, Velusamy et al.'s Integral Miss Controller (IMC), and Zhou et al.'s Adaptive Mode Control (AMC).

Per-line adaptive cache decay uses a per-cache-line adaptive scheme, whose implementation involves counters as in baseline cache decay. Each line selects one decay interval among a choice of 10 different values using a multiplexer. After a line is decayed, the counter is used to gauge the time of the next access. If the next miss occurs very early (when the counter has not even reached a quarter of its maximum value), then the miss is assumed to be an induced miss and the decay interval is increased by a factor of two. If the miss occurs late (after the counter has crossed three-quarters of its maximum value), it is assumed to be a miss to a dead line and the decay interval is decreased by a factor of two. A miss occurring at other times causes no change to the decay interval.

IMC and AMC are global schemes which adapt the decay interval at the whole cache granularity. Further, instead of using a counter, they keep the tag array powered on and determine if a miss is induced. AMC tries to track the default program behaviour without decay by monitoring the actual miss rate and the induced miss rate. It forces the induced miss rate to track the actual miss rate within a margin of a fixed threshold. This threshold is called its Performance Factor (PF). IMC takes the approach of formal feedback control in adapting the decay interval. The set-point to the controller is given in terms of induced misses per cycle which the controller tracks by adapting the decay interval.

5 Results

This section presents the results of our evaluation of the aforementioned techniques. We compare our technique to an omniscient selection of individual-best decay intervals and to previously proposed adaptive techniques.

5.1 Accuracy of estimation

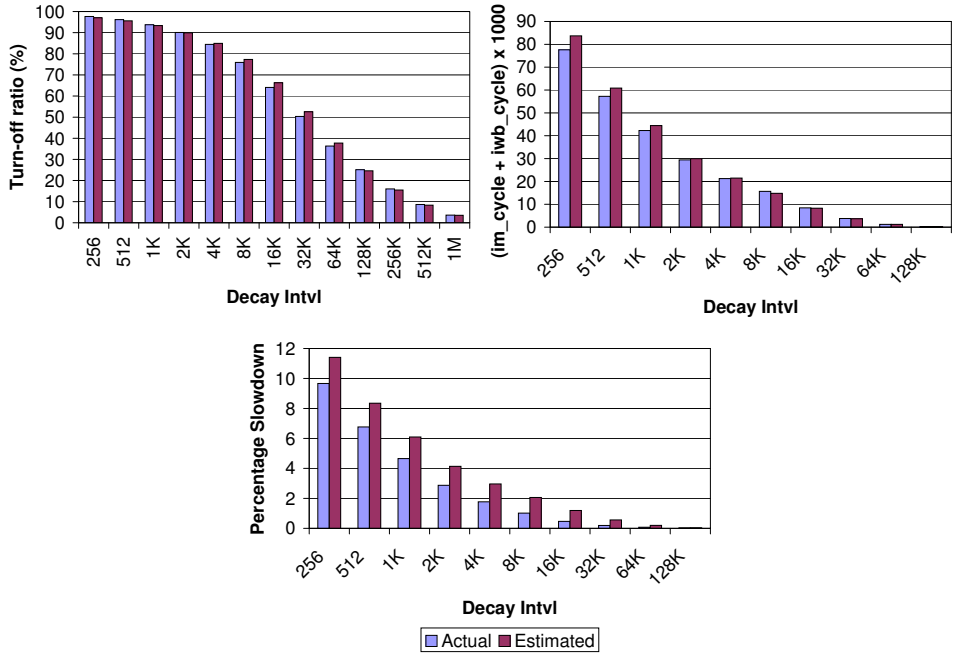

Figure 1. (a) Turn-off ratio (b) *im_cycle* **+** *iwb_cycle* **and (c) Percent slowdown for a typical benchmark—gcc. Left bar indicates Actual values and right bar indicates Estimated values.**

Before we compare our technique to other techniques, it is first essential to ascertain its prediction accuracy. This helps us both to verify the validity of the analysis described in Section 3 and to understand which of the sources of inaccuracies

mentioned there (performance slowdown, replacement policy etc.) are substantial with respect to the choice of a decay interval.

As a candidate, we study a typical benchmark, gcc. It is to be noted that unlike the experiments evaluating the effectiveness of the various adaptation techniques, here we use the same reference input set, both for profiling and actual measurement. This is just to understand the accuracy of our estimation method. Figure 1 shows the results of the profiling. It can be seen that the estimated values of tor and $im\_cycle + iwb\_cycle$ track their actual values very closely, while those of *perc_slowdown* are offset by a factor. The Root Mean Square (RMS) values of the percentage error in the first two cases are 2.4% and 4.2% respectively, indicating that the equations in Section 3 estimate those parameters with a reasonable degree of accuracy. However, the performance degradation graph shows a larger difference between the actual and estimated values. This is because, the heuristic value of the effective latency (2 cycles for integer benchmarks) is higher than the actual value for gcc. Since the performance degradation also depends on the *im_cycle* number, which is estimated more accurately using analysis, the estimated values of performance degradation show the same *trend* as the actual values. This is also further substantiated by the fact that the correlation co-efficient between the actual and estimated performance degradation is high at 99.5%.

From the graphs above, it can be concluded that the estimation method performs as expected with a reasonable degree of accuracy. In other words, the factors unaccounted for in the equations of Section 3 viz., eager write-backs, change in replacement behaviour and elongation of dead times and access intervals during decay do not impact the accuracy of estimation substantially. Further, the heuristic calculation of the effective latency leads to inaccuracy in the estimation of performance degradation while preserving the shape of the trend. The fact that the actual value of the best decay interval for gcc (16K cycles) matches the estimated value for the above mentioned experiment shows that the *tracking* behaviour of performance degradation overweighs its estimation inaccuracy.

5.2 Prediction of the optimal decay interval

Now, we evaluate how well our profiling technique predicts the optimal decay interval. Profiling is done for the benchmarks using the train input set and the predicted optimal value for the decay interval is compared against the actual optimum obtained by simulating cache decay using the reference input set. The prediction is done based on two metrics: *esav* and ED^2. The former method is called *esav-pred* and the latter as *ed2-pred*. The real optimum is computed by running the benchmarks with different decay intervals ranging from 256 to 1M cycles and choosing the best interval based on the corresponding desired target metric (*esav* or ED^2). Table 4 shows the optimal decay intervals for these methods. The *esav-best* and *ed2-best* columns show the actual optima. Values that match are shown in bold face font. Clearly, ED^2 favours larger decay intervals since it more heavily emphasizes performance loss. Also, the best ED^2 point lies to the right of the best *esav* point. Qualitatively, it can be seen that the *esav-pred* scheme tracks the optimal decay interval closer than the *ed2-pred* scheme. In spite of the number of mis-predictions being greater in *esav-pred*, it is off from the actual value at most by a factor of 4 while the *ed2-pred* technique is off by up to a factor of 16. However, the average number of power-two steps by which these techniques are off the actual optima is the same for both the schemes (0.7). So, it is not quantitatively clear from this data, which of them is more accurate. This is because, the observed effect is a compounded result of cross-training and inaccuracy of estimation. Isolating these will help us understand the accuracy of estimation better.

In order to quantitatively ascertain which of the above two techniques is more accurate and in order to understand how much of the mismatch observed in Table 4 is due to cross-training as opposed to inaccuracy in estimation, we also compare the optimal decay interval values obtained by self-training to the actual optima. That is, we use the same reference input set, both for profiling and cache decay simulation process. For such an experiment, the number of mis-predictions drops from 15 to 7 for *esav-pred* and from 12 to 8 for *ed2-pred*. Similarly, the average number of steps by which the prediction is off, goes from 0.7 to 0.3 for *esav-pred* and to 0.42 for *ed2-pred*. Thus, it can be seen that about half the mis-match in the techniques, which is a not-so-neglibigle portion of the inaccuracy, stems from cross-training. This also clearly shows that *esav-pred* is more accurate when compared to *ed2-pred*. This is expected because, the heuristic estimate of delay plays a more important role in ED^2 metric than in the *esav* metric.

5.3 Overall performance of the schemes

In this section, we present the results of our evaluation of the various schemes. We compare the different adaptive schemes against an omniscient choice of individual-best decay interval with respect to energy savings. This oracle scheme is called *esav-best*. We also evaluate a non-adaptive scheme which sets the decay interval to a constant value of 4K cycles, which is the single best constant decay interval in terms of energy savings. This static technique is called *esav-avg*. Our profile-

Table 4. Optimal decay interval—actual and predicted values for *esav* and ED^2 methods.

bench	esav-best	esav-pred	ed2-best	ed2-pred
INT				
mcf	1K	2K	2K	32K
gap	2K	8K	**32K**	**32K**
vpr	4K	8K	16K	32K
vortex	4K	16K	16K	64K
bzip2	**4K**	**4K**	16K	8K
twolf	**4K**	**4K**	16K	32K
gzip	**8K**	**8K**	**64K**	**64K**
perlbmk	8K	16K	64K	256K
gcc	**16K**	**16K**	32K	64K
crafty	**16K**	**16K**	64K	128K
parser	16K	8K	**64K**	**64K**
eon	**16K**	**16K**	**64K**	**64K**
FP				
art	**256**	**256**	**256**	**256**
swim	1K	2K	512	2K
lucas	**1K**	**1K**	**1K**	**1K**
wupwise	2K	8K	**8K**	**8K**
applu	**2K**	**2K**	**2K**	**2K**
equake	2K	4K	**4K**	**4K**
facerec	2K	4K	**8K**	**8K**
mesa	4K	2K	8K	4K
galgel	4K	2K	4K	2K
ammp	4K	2K	**8K**	**8K**
fma3d	**4K**	**4K**	**4K**	**4K**
mgrid	8K	4K	8K	4K
sixtrack	**16K**	**16K**	**16K**	**16K**
apsi	16K	8K	**16K**	**16K**

based decay interval estimation is called *esav-pred*. In order to understand the impact of cross-training, we also consider the self-training version of *esav-pred*, called *esav-self*, where both the profiling and measurement are done with same input data set. The evaluation also includes the profiling scheme based upon the ED^2 metric. This is called *ed2-pred*. This chooses the best ED^2 point from its estimation with the rationale being that DVS can be applied over and above cache decay to achieve the energy target. In order to be able to compare the energy savings between the ED^2 and *esav* methods, we use a DVS setting that is computed a priori. The DVS setting is chosen as follows: for a decay interval chosen by *ed2-pred*, we let the tolerable performance degradation to be the one that would be seen by the corresponding *esav* scheme, which is *esav-pred*. The voltage is scaled enough to keep the performance degradation within this limit, thereby saving energy without losing much performance. Since DVS leads to quadratic energy savings and since the chosen DVS setting is applied over and above cache decay, the expecation is that the combination could be more energy efficient than the respective individual techniques (DVS and cache decay) themselves. This resultant energy saved is the one compared against that of the *esav* schemes. Since this DVS setting cannot be computed without actually executing the application, this experiment is does not aim at a realistic implementation. It is just used as a study to understand the potential of the scheme.

The various adaptive schemes we evaluate are: Integral Miss Controller (called *imc* in the graphs), Adaptive Mode Control (called *amc* in the graphs), and per-line counter-based adaptive decay. The set-point for IMC was calculated offline from the induced misses per cycle data for the average-best decay interval of 4K cycles. The average number of induced misses per cycle across all benchmarks was 0.005, hence we used it as the set-point for the controller. Similarly, for AMC, we chose the Performance Factor (PF) value that maximizes the average energy savings across all applications. We found this value to be 0.25. Since IMC and AMC keep their tags always powered on to measure the induced misses, we had to factor in the cost of doing so in our energy calculations. For our L1 data cache configuration, we used the HotLeakage 1.0 [22] and CACTI 3.2 [17] tools to estimate the ratio of tag array power to that of the data array for a 130 nm process. We found this ratio to be about 11%. Hence, while computing the normalized leakage energy savings (*esav*), we scale down the turnoff-ratio for these techniques commensurate to this ratio. Also, to understand the potential of the adaptive schemes to adapt to the optimum decay interval, we consider two more ideal schemes, *imc-ideal* and *amc-ideal*. These assume that the induced misses can be measured magically without keeping the tags powered on always. Studying these two ideal schemes helps us understand the cost-benefit trade-off in keeping the tags on versus online adaptivity.

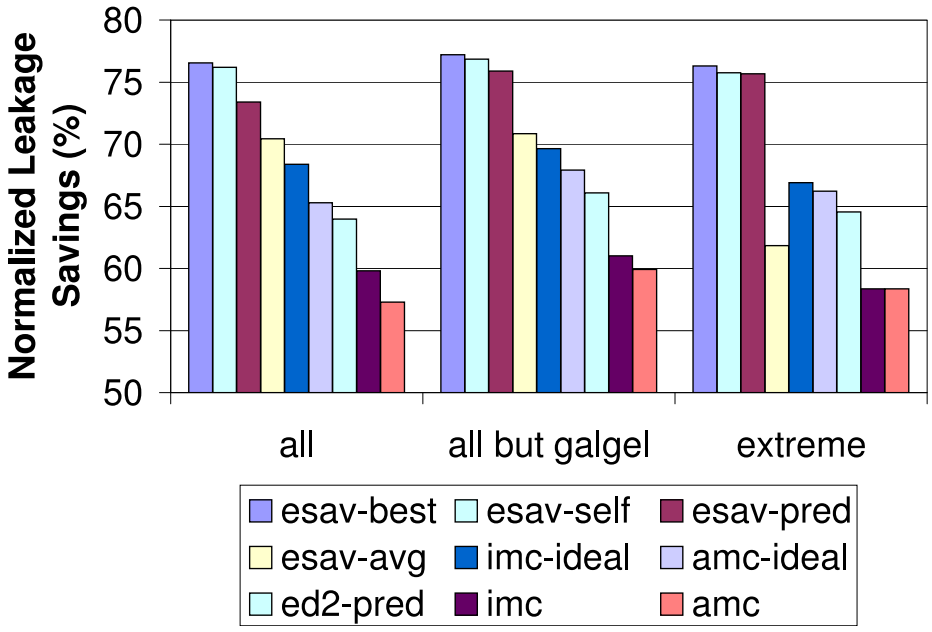

Figure 2. Normalized leakage energy savings for the various decay interval adaptation schemes. The leftmost group of the graph shows averages over all benchmarks. The middle group shows averages over all benchmarks except galgel. The rightmost group shows averages over the ten extreme-case applications.

Figure 2 shows the normalized leakage energy savings (*esav*—the percentage of total cache leakage energy saved) for the above mentioned decay schemes in the order of their performance. First, the absence of per-line counter-based adaptive cache decay is conspicuous in the graph. It is not included in the graph because per-line adaptive decay selects overly agressive decay intervals which lead to much higher slowdowns when compared to other schemes. While the average slowdown of other schemes is less than 1.2% of the baseline execution time, the average for per-line adaptive decay is 4.4%. This leads to its poor performance both in terms of energy and delay. This behaviour is also reported in [23]. The normalized leakage energy savings for this scheme averages around 3.2% of the total cache leakage energy, which is much smaller than the savings obtained by other schemes. Our belief is that this problem arises due to lack of sufficient hysteresis in the two-bit counters to be able to filter out the noise in the access interval pattern. Hence, per-line adaptive decay has been omitted from the graph above and will not be included in further discussions below.

Graph in Figure 2 shows three groups. The leftmost group shows averages over all SPEC2000 benchmarks. The middle group shows averages over all benchmarks except galgel. We observe that galgel has very different access pattern during profiling and measurement. The decay interval predicted by *esav-pred* (2K cycles) is off from the optimal value (4K cycles) by just one step. This amounts to very minimal difference in energy savings in the profile data but leads to a substantial difference of 50% in the reference execution. This difference between profiling and measurement shifts the average by a substantial value in a way that impacts the profile-based schemes negatively. In order to show this cross-training behaviour, we indicate the data from all benchmarks except galgel seperately. The results of the *esav-self* technique also illustrates this behaviour clearly. It can be seen that the self-training results for *esav-pred* scheme (*esav-self*) matches closely with the cross-training results unless galgel is included.

The rightmost group shows averages only amongst those benchmarks whose optimal decay interval is further from the average-best decay interval. Since 4K is the average-best decay interval, we chose to exclude benchmarks with 2K, 4K and 8K cycles as their optimal points. This is done to highlight the adaptive behaviour of our profile-based technique. These 'extreme' applications are the main motivation behind adaptive techniques like ours. A good adaptive technique should perform at least as well as a non-adaptive scheme in the average case and should perform much better in the extreme cases. Our *esav-pred* scheme performs slightly better in the average case and much better in the extreme cases. The extreme cases are shown separately to emphasize this point.

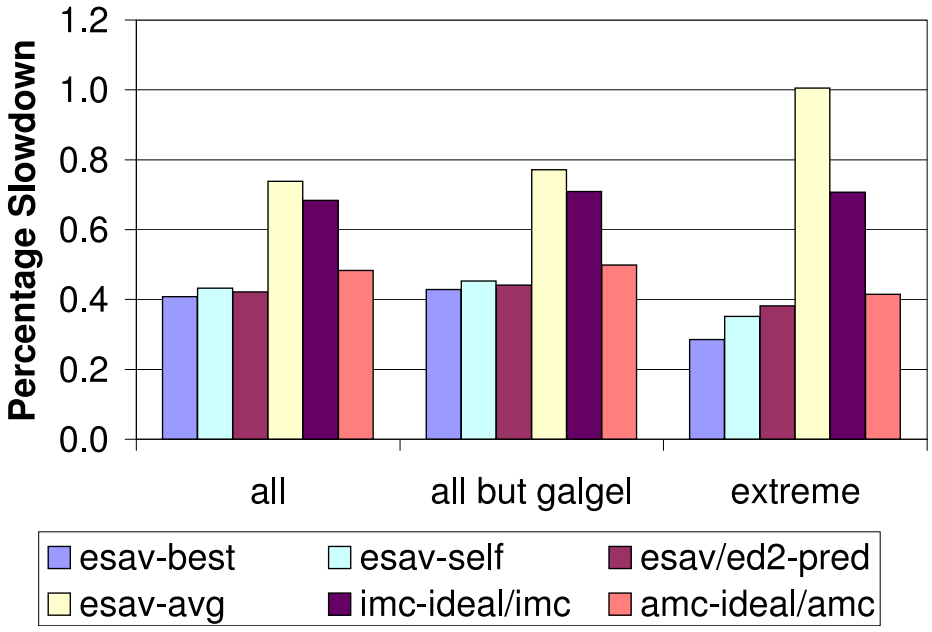

Figure 3. Percentage slowdown for the various decay interval adaptation schemes.

Figure 3 also shows data about the various decay schemes. It shows the percentage slowdown of the various schemes. Keeping with the convention above, this graph is also divided into three groups. Since *ed2-pred* uses the performance degradation of *esav-pred* as its tolerable limit, both *esav-pred* and *ed2-pred* are denoted by the same bar (*esav/ed2-pred*).

From the graphs above, it can clearly be seen that the ideal scheme *esav-best* and the self-training profile-based scheme

(*esav-self*) perform better than the rest. It can also be seen that *esav-pred* technique performs better than all of the non-omniscient schemes. Also, it is close to the omniscient *esav-best* scheme for the most part. Though on an average, it bridges only about half the gap (3% of the total cache leakage) between *esav-avg* and *esav-best*, it is almost as good as *esav-best* for most of the applications. As the middle graph in Figure 2 shows, out of the maximum possible improvement of 6.1% of the total cache leakage, it achieves about 5%, falling short only by 1.1%. Also, it performs a lot better in the extreme cases, improving upon *esav-avg* by 14%. It should also be noted that if we had the luxury of self-training, *esav-pred* performs virtually equal to the omniscient *esav-best* scheme. It is also worth observing from Figure 2 that the first three bars remain almost constant across the three groups. Since they perform consistently across applications, this shows the robustness of those schemes. Moreover, from the slowdown graph in Figure 3, it can be observed that the slowdown due to all the above decay techniques is minimal (less than 1.2%).

The energy savings graph also shows that the profile-based ED^2 scheme (*ed2-pred*) performs poorly in spite of being an ideal potential study. This is primarily due to our heuristic estimation of performance degradation. The effect of cross-training magnifies this inaccuracy. Since *ed2-pred* weighs performance heavily, it is impacted by the inaccuracy in the estimation of performance degradation worse than *esav-pred*. Furthermore, it is evident from the graph that both the ideal and non-ideal dynamic adaptation schemes do not perform as well as *esav-pred* or even *esav-avg*. In case of AMC, this is mainly because of failure in adaptation. The average decay interval during the execution of the program for AMC is off the static optimal value by about 2.1 steps on the average. While IMC is accurate in this respect (better than *esav-avg* and even *esav-pred*), its poor performance is because of noise and phase difference in adaptation. By the time IMC adapts to a decay interval, the current optimum changes. Moreover, it goes into periods of oscillation when a decay interval chosen is too small (or large) while its next online choice is too large (or small). This results in its alternating between the two sub-optimal intervals even while the average of the two is close to optimum. This mainly happens when the decay interval is close to mid-range. This is evident from figure 2 because the ideal versions of both IMC and AMC, perform better than *esav-avg* for extreme applications but do worse for the others. The high level of noise inherent to access interval patterns and the difficulty in tuning their parameters are further problems in these online decay adaptation schemes. The performance factor (PF) of AMC and the set-point for IMC are difficult parameters to tune. They vary depending on the application behaviour. This is where a profile-based scheme like ours wins. Also, an offline scheme or a scheme that operates at much coarser granularities (like program phases etc.) filters out the noise through large-scale averaging. Apart from these problems in adaptation, the non-ideal versions of these schemes are also affected by the cost of keeping their tags on. This is the reason why in spite of their ideal versions performing better than *esav-avg*, the non-ideal versions perform poorly in case of extreme applications.

5.4 Effect of associativity

It was pointed out in Section 3 that a difference in replacement behaviour arises between profiling and actual measurement of decay. Our equations in Section 3 do not take this into account. Hence, we explore the impact of this change in replacement behaviour on the effectiveness of our profile-based schemes. The chance for a change in replacement behaviour during decay increases with the associativity of a cache. So, we ran experiments evaluating the various decay adaptation schemes by changing the assosciativity of the 2-way cache used in previous evaluations to 1-way and 4-way. Figure 4 shows the results of this study for the non-ideal schemes and *esav-best*. It is to be mentioned that for the adaptive schemes (IMC and AMC) we chose the parameters as before. The average number of induced misses per cycle for the single-best decay interval of 4K cycles were 0.009 and 0.0045 respectively for the 1-way and 4-way cache configurations. These were chosen as the respective set-points for IMC. For AMC, a PF value of 0.25 was found to be the best in terms of average energy savings for all the cache configurations. Hence, we used the same value of 0.25 for all the caches.

It can be seen that the general trend in the order of performance of the various schemes remains the same across different associativities. This confirms the observation made in Section 5.1 that the change in replacement behaviour is not a significant source of inaccuracy. Also, the energy saved increases with the associativity. This is because the extra ways hold more dead lines and provide more opportunity to put them into standby mode. The slowdown observed in this experiment also shows similar trends as before. All the schemes show very minimal slowdowns. Further, the slowdowns decrease with increasing associativity. The average slowdown for all the schemes is at most 2.5% of the baseline execution time for the 1-way cache and 1.1% for the 2-way cache while it drops to 0.5% for the 4-way cache.

The results and the discussions presented above show that the profile-based decay adaptation scheme *esav-pred* performs almost as well as an omniscient choice of per-benchmark decay interval. The scheme is also robust in terms of adaptation, performs reasonably well in the average case and is vastly superior for the extreme-case applications.

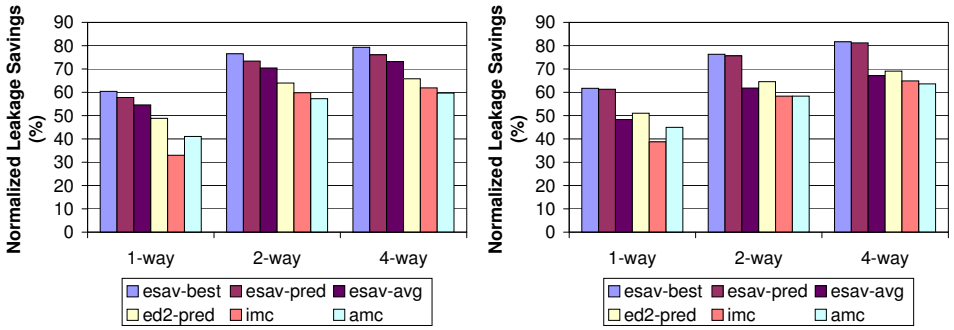

Figure 4. Normalized leakage savings with different cache associativities for (a) all benchmarks and (b) only the extreme-case applications.

6 Conclusion and future work

This paper introduces a profile-based adaptive scheme for cache decay. The key insight behind this work is that the most important variables determining the energy saved in a decay mechanism can be estimated from analyzing profile data. The decay interval chosen by such an analysis matches the optimal decay interval with a reasonable degree of accuracy. This makes a profile-based scheme that optimizes for normalized leakage energy savings ('esav-pred') better than the dynamic adaptive schemes and *almost as good* as an omniscient choice of individual-best decay interval. The 'esav-pred' scheme performs better than online adaptive techniques because of their failure in robust adaptation and the cost of keeping the tags powered on. It is also difficult to tune their parameters, especially in the face of noisy access-interval patterns. In contrast, 'esav-pred' chooses the decay interval best for the program, eliminating noise in the cache access pattern and without having to keep the tags powered on. Hence, the profile-based decay adaptation scheme 'esav-pred' achieves its adaptation target successfully: it performs slightly better than the average-best case for most benchmarks and much better than the average-best case for the extreme-case applications.

We also find that the primary source of inaccuracy in the estimation of the optimal decay interval arises from the heuristic calculation of the effective latency of the cache. This is also the main reason why the *ed2-pred* scheme based on the ED^2 metric performed poorly. It heavily emphasizes the performance factor by squaring the delay factor. This work also shows that the luxury of self-training could improve the potential of the profiling scheme even more. This makes it interesting to explore the effect of extending the analysis described in this paper to dynamic adaptation as opposed to static profiling. Such a dynamic scheme will have the advantage of sampling the actual input data set itself.

The results of this paper show not only the importance of adaptation to program behaviour but also the opportunities afforded by profiling in the context of energy efficiency. Similar to [9], we find that our profile-based optimization performs better than hardware-based adaptation schemes (IMC and AMC).

Future work in this direction could explore the estimation of performance degradation. An analytical estimate of the effective latency of programs will not only be useful to the profile-based schemes described in this paper but also the profiling community in general. The choice of the decay interval matters for state-preserving techniques too, only to a lesser degree. So, the extension of this study to state-preserving leakage saving techniques like drowsy cache [4] is one more area for further exploration. Also, extension of this work to make the scheme dynamically adaptive by combining it with phase detection research is another interesting venue for future research. A detailed sensitivity analysis on the constants (parameters) we have used in this work to better understand their impact on the leakage energy savings is a further aspect that one might explore. Application of this technique to the instruction cache, lower level caches and other structures within the CPU is another fruitful direction for future work. Extension of this work to the level of per-line adaptation could provide interesting insights on cache behaviour. Furthermore, studying the implications of the profiling scheme in the context of SMT and CMP is another potential future direction.

Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. CCR-0133634 and CCR-0105626. We would also like to express our sincere thanks to Mircea Stan for his helpful and constructive feedback.

References

[1] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: A transparent dynamic optimization system. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Languages Design and Implementation*, June 2000.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[3] D. C. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, University of Wisconsin-Madison Computer Sciences Department, July 1996.

[4] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 147–57, May 2002.

[5] M. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: Application to energy reduction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 157–168, Jun. 2003.

[6] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[7] T. Kobayashi and T. Sakurai. Self-adjusting threshold-voltage scheme (sats) for low-voltage high-speed operation. In *Proc. IEEE 1994 CICC*, pages 271–274, May 1994.

[8] H. Lee, G. Tyson, and M. Farrens. Eager writeback - a technique for improving bandwidth utilization. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 11–21, Dec. 2000.

[9] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 14–27, Jun. 2003.

[10] K. Nii et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 293–98, Aug. 1998.

[11] D. Parikh, Y. Zhang, K. Sankaranarayanan, K. Skadron, and M. Stan. Comparison of state-preserving vs. non-state preserving leakage control in caches. In *Proceedings of the 2003 Workshop on Duplicating, Deconstructing, and Debunking*, June 2003.

[12] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 90–95, July 2000.

[13] K. Sankaranarayanan and K. Skadron. Profile-based adaptation for cache decay. *ACM Transactions on Architecture and Code Optimization*, Sep. 2004. To appear.

[14] K. Scott, N. Kumar, S. Velusamy, B. Childers, J. W. Davidson, and M. L. Soffa. Retargetable and reconfigurable software dynamic translation. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 36–47, Mar. 2003.

[15] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, Oct. 2002.

[16] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 336–349, Jun. 2003.

[17] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical Report 2001/2, Compaq Western Research Laboratory, Aug. 2001.

[18] SIA. *International Technology Roadmap for Semiconductors*, 2001.

[19] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13, Apr. 2003.

[20] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*, May 2002.

[21] 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. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, Feb. 2001.

[22] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia Department of Computer Science, Mar. 2003.

[23] H. Zhou, M. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: A static-power-efficient cache design. *ACM Transactions on Embedded Computing Systems*, 2(3):347–372, 2003.

[24] V. Zyuban. Unified architecture level energy-efficiency metric. In *Proceedings of the 2002 Great Lakes VLSI Symposium*, pages 24–29, Apr. 2002.