

Profile-Based Adaptation for Cache Decay

Karthik Sankaranarayanan
Department of Computer Science
University of Virginia, Charlottesville, VA
MCS Project Report, July 2003

Abstract

Cache decay is a leakage-reduction mechanism that puts 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 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 using profiling with a reasonable degree of accuracy. Unlike previous methods that attempt to put only ‘dead’ lines into standby mode, this work explicitly trades off the leakage power saved in putting ‘live’ lines into standby against its performance and energy costs. It also combines traditional DVS with cache decay using an ED^2 analysis and obtains results 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 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 [15] 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 [10]. 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 offline analysis to determine the optimal value of the decay interval on a per-application granularity for maximizing energy savings subject to a constraint of very little or no performance loss subject to an ED^2 optimization framework [20], which is voltage-independent.

Related Work. Kaxiras *et al*’s work on cache decay [6] improves upon a coarser-granularity work by Yang *et al* [17] on instruction caches which shuts down groups of I-Cache lines. Both these techniques use the *gated- V_{dd}* technique [11] 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, 9] and drowsy caches [4] have been proposed, but may be inferior for L1 caches that are backed by fast, on-chip L2 caches [10]. 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* [18] and Velusamy *et al* [16]. Kaxiras *et al*. [6] 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 whole cache granularity. 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 perform the adapta-

tion. While both Kaxiras *et al.*'s work and AMC try to minimize the performance loss, they don't explicitly consider the tradeoff of shutting down live lines. IMC indirectly considers the shutting down of live lines through its choice of setpoint but suffers from the non-adaptation of the setpoint. 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 vs delay tradeoff of shutting down live lines. It also uses ED^2 analysis to combine traditional DVS with cache decay, thereby performing 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.

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 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 vs. 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. On the other hand, if the decay interval is too large, many dead lines would not be put into standby, hence dissipating 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 argument to obtain a constant 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.

2.2 Motivation

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. Figure 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 figure shows statistics about access intervals. The second figure 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 third figure 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 third figure 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 a fractional (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 tradeoff of shutting down live lines, 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 tradeoff of putting live lines into standby mode. In order to consider such a tradeoff, 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 also perform a voltage-independent ED^2 analysis [20] using the estimated variables and combine cache decay with traditional DVS in order to enhance the energy savings of the profile-based scheme. Such an enhanced profile-based scheme performs almost equal to a premonitory selection of the individual best decay interval.

	avg	stdev	max		avg	stdev	max		ideal tor	best intvl
art	13	27	4092	art	3965	6443	42544	art	99.02	256
twolf	254	2515	304701	twolf	29270	23070	457980	twolf	87.21	4096
eon	751	102647	380539664	eon	68002	1877187	483994770	eon	10.93	16384

Figure 1. (a) Access interval statistics (b) Dead time statistics and (c) Best decay interval and oracle turn-off ratio.

3 Methodology

This section describes in detail, the profile based estimation technique and the ED^2 analysis 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 lower level cache due to the turning off of 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 writeback. If this writeback is an *extra* writeback, which was not present in the default program behaviour without decay, it is called an *induced writeback* (iwb). It is to be noted that writebacks that occur during the decay of a live line need not always be induced. They could also be eager writebacks [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 [16] and AMC [18] additionally, this 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, *idling* energy is spent during the extra cycles the program takes to complete. Assuming ideal clock gating, this primarily arises from the leakage energy in the rest of the processor.

Amongst these four variables, the energy due to overhead hardware for cache decay has been known to be negligible [6] and hence can be omitted. However, it cannot be omitted for AMC or IMC since the tag array dissipates non-negligible fraction of the total cache power.

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 writebacks divided by the baseline program running time in cycles respectively. $perc_slowdown$ 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 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. 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 extra 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 mandatory in reporting energy savings, 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 - 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 $t_1 \dots t_n$ 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 we assume decay intervals

which are powers of two like Kaxiras *et al.* 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 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 $a(t_i) + a(t_{i+1}) + \dots + a(t_n)$. 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,

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

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 while dead lines lead 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 *misprediction 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 writebacks are the *mispredictions*. Hence, with a reasonable degree of approximation, *iwb* can be obtained by multiplying this by the misprediction 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 doesn't account for eager writebacks. 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 running 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 *mid-point* lies at $4K\sqrt{2}$. The *time contribution* of the dead times under consideration is $1000 \times 4K\sqrt{2}$. Similarly, the time contribution of the live times is $10 \times 4K\sqrt{2}$.

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 $a(t_i) \times t_i \times \sqrt{2}$. 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 $AA(t_i)$, then, the total standby time contribution due to switching live lines into standby mode is given by $AA(t_i)\sqrt{2}$. Similarly, if we denote $\sum_{j=i}^n t_j d(t_j)$ by $DD(t_i)$, then the total standby time contribution due to switching off of dead lines is given by $DD(t_i)\sqrt{2}$. 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

$$(AA(t_i) + DD(t_i))\sqrt{2} - t_i(A(t_i) + D(t_i))$$

. 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.,

$$tor(t_i) = \frac{(AA(t_i) + DD(t_i))\sqrt{2} - t_i(A(t_i) + D(t_i))}{n_lines * n_cycles} \quad (4)$$

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} . However, we find that this need not be true. In order to deal with the non-uniform behaviour, we make use of the fact that $tor(0) = 1$. In order that it is true, we calculate the multiplication factor that takes the place of $\sqrt{2}$ and use it in our estimation. This approximates the skewed distribution within an interval in a heuristic manner.

From the equations (especially one for *tor*) above, 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 it is beneficial to shut down live lines for a particular application. This is something that most other decay interval adaptation techniques lack. Closing down of live lines comes as a *side-effect* in them. It is not an explicit design point that is traded off.

The fourth and final variable in the *esav* equation is the performance 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 go to the lower level. This *effective latency* is not the normal latency to the lower level - it is much lesser. Most of the normal latency is hidden by the 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 are 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 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, 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 with the actual optimum with a reasonable degree of accuracy. Hence, the equation we use for performance degradation is

$$perc_slowdown(t_i) = eff_lat * im_cycle(t_i) \quad (5)$$

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

1. We do not account for eager writebacks directly in the analytical expressions.
2. Since the decay process slows down the program, few access intervals and dead times would be 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 behaviour. Although the line in standby mode is not LRU, it will be the one to get evicted first.

Since the events associated with these effects would 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 analysed

in the same way 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.

An alternative to choosing the best *esav* point is to go for the best ED^2 point [20]. Since DVS combined with frequency scaling can be applied to most scenarios, any power saving technique should atleast be as good as DVS. 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 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 can't 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 [15], this is reasonable to assume for technologies beyond 130 nm.

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 technique may not perform well. We find that in terms of estimation accuracy, it does worse than the *esav* technique. But with the right choice of DVS setting, it is capable of bridging the very small gap between the *esav* technique and the omniscient individual-best decay interval selection.

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. 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 useful as 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, 14] could be used to select different profiling points and the dynamic translation infrastructure (eg. [12]) 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 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 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*.

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

Table 1. Configuration of simulated processor microarchitecture.

The microarchitectural configuration we use to evaluate the techniques resembles Alpha 21364 as closely as possible. Table 1 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 2. 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 behavior. For each program, we use the reference input set and fast-forward to a single representative sample of 1 billion instructions. The location of this sample is chosen

using the data provided by Sherwood *et al* [13]. Simulation is conducted using SimpleScalar’s EIO traces to ensure reproducible results for each benchmark across multiple simulations. It is to be noted that we use the same data for profile-based estimation and evaluation. We did attempt cross-training but a few benchmarks indicated non-representative behaviour and hence chose self-training. We understand that this self-training is a matter of concern in the research community and would address this issue by finding an alternate representative input set as soon as possible.

bench	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

Table 2. Summary statistics used for the benchmarks in this study. All benchmarks use reference inputs.

4.3 Other adaptive techniques

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

Adaptive cache decay uses a per-cache-line adaptive scheme whose implementation involves counters as in base-line cache decay. Each line selects one decay interval

among 10 choices using a multiplexer. After a line is decayed, the counter is used to gauge the time of 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 $\pm 50\%$. IMC takes the approach of formal feedback control in adapting the decay interval. The setpoint 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

Before we compare our technique to other techniques, it is first essential to ascertain its accuracy. It 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. Figure 2 shows the results of the study. It can be seen that the actual and estimated values are very close, indicating that the estimation technique is valid. Further, it is clear that the differences in the numbers due to extra running time, change in replacement policy etc. are not significant. It is to be noted that the performance degradation graph also shows a similar *tracking* behaviour. This is in spite of the heuristic estimate of the effective latency. The reason is because the heuristic value of the effective latency (2 cycles for integer benchmarks) is close to the actual value for gcc. Further, performance degradation also depends on the *im_cycle* number, which is computed from the analysis and not heuristically.

5.2 Prediction of optimal decay interval

Table 3 shows the optimal decay intervals based on *esav* and ED^2 metrics. Clearly, ED^2 favours larger decay inter-

bench	esav-best	esav-pred	ed2-best	ed2-pred
INT				
mcf	1K	1K	2K	2K
gap	2K	2K	32K	32K
vpr	4K	4K	16K	32K
vortex	4K	8K	16K	64K
bzip2	4K	8K	16K	32K
twolf	4K	4K	16K	16K
gzip	8K	8K	64K	64K
perlbnk	8K	8K	64K	64K
gcc	16K	16K	32K	128K
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	2K	8K	8K
applu	2K	2K	2K	4K
equake	2K	2K	4K	4K
facerec	2K	4K	8K	8K
mesa	4K	8K	8K	8K
galgel	4K	4K	4K	4K
ammp	4K	4K	8K	8K
fma3d	4K	4K	4K	8K
mgrid	8K	8K	8K	8K
sixtrack	16K	16K	16K	16K
apsi	16K	4K	16K	16K

Table 3. Optimal decay interval - actual and predicted values for *esav* and ED^2 methods

vals since the best ED^2 point lies to the right of the best *esav* point. Also, both the schemes track the optimal decay interval trend reasonably well. Both the actual (*esav*-best, *ed2*-best) and the predicted values (*esav*-pred, *ed2*-pred) are shown. It can be seen that ED^2 prediction is worse compared to *esav*. The average error ratio, which is the ratio of the larger decay interval to the smaller among the predicted and actual values, is 1.35 for *esav* and 1.63 for ED^2 - i.e., ED^2 is off its optimal value by a factor that is greater than the one for *esav*. Further, the number of wrong choices (of optimal decay interval) made by *esav* prediction (7) is lesser than that made by the ED^2 method (8). It can also be seen that except for *apsi*, *esav* method is not off the optimal value by more than a factor of two. Hence, *esav* method is more accurate when compared to ED^2 method. 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

Here, we present the overall performance results of the various schemes. Figure 3 shows the normalized leakage energy savings and slowdown data for the various schemes. In the graphs, 'esav-best' is the omniscient individual-best

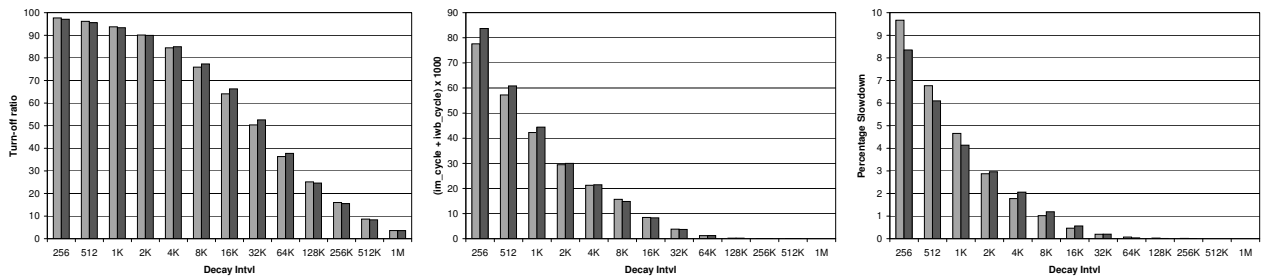


Figure 2. (a) Turn-off ratio (b) $im_cycle + iwb_cycle$ and (c) Percent slowdown for a typical benchmark - gcc. Left bar indicates Actual values and right bar indicates Estimated values.

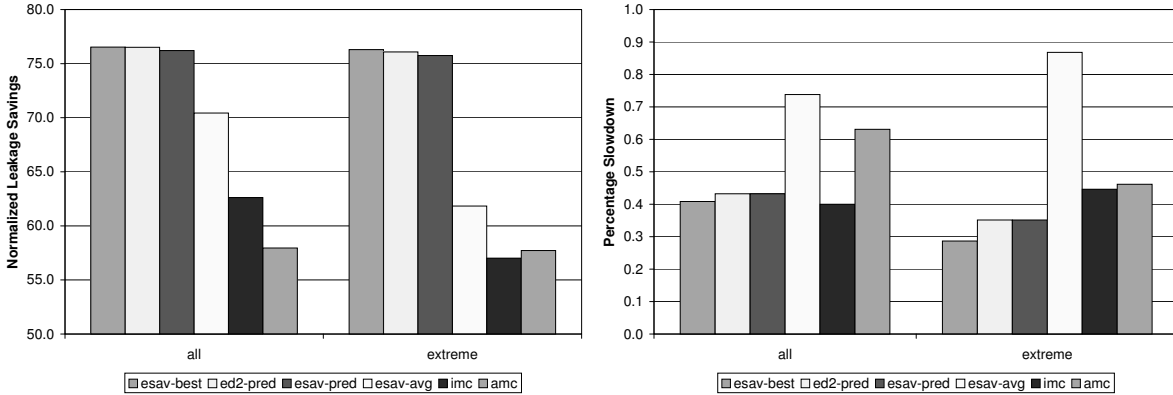


Figure 3. (a) Normalized leakage energy savings and (c) Percent slowdown for the various decay interval adaptation schemes. Left portion of the graph shows averages over all benchmarks while the right portion shows averages over the ten 'extreme' applications.

choice of decay interval. 'esav-pred' is our *esav* scheme. 'ed2-pred' is a demonstrative scheme. It uses ED^2 estimator to pick its decay interval. However, to reach the optimal point with the ED^2 scheme, we also need to find the level of DVS setting to be applied. Here, we assume that the optimal DVS setting is given apriori. This serves as a 'potential study' to help us understand the capability of the ED^2 scheme. The apriori DVS setting is chosen by letting the tolerable performance degradation to be the one seen by 'esav-pred'. The DVS setting scales the voltage in such a way that the performance of 'ed2-pred' is the same as 'esav-pred'. However, due to quadratic energy savings, this would be more energy efficient. 'esav-avg' is constant decay where we pick a single average-best decay interval for all the benchmarks. We find that this average-best interval is 4K cycles and use that for 'esav-avg'. 'amc' is Adaptive Mode Control and 'imc' is Integral Miss Controller. The setpoint for IMC was calculated offline from the induced misses per cycle data for the average-best decay interval. The average number of induced misses per cycle across all benchmarks was 0.005, hence we used it as the setpoint for the controller. The absence of adaptive cache decay is conspicuous in the graphs. It is not included

in the graphs because adaptive decay selects overly aggressive decay intervals which lead to higher slowdowns when compared to other schemes. While the average slowdown of other schemes is less than 1%, the average for adaptive decay is 4%. This leads to its poor performance both in terms of energy and delay. This behaviour is also reported by Zhou *et al* in [19]. 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 adaptive cache decay has been omitted from the graphs above.

Each graph shows two portions. The left portion shows averages over all SPEC2000 benchmarks. The right portion 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 throw light on the adaptive behaviour of our technique. These 'extreme' applications are the main motivation behind adaptive techniques like ours. A good adaptive technique should perform atleast as well as a non-adaptive scheme in the average case and should perform much better in the extreme cases. Our

scheme even does better - it performs slightly better in the average case and performs much better in the extreme cases. The extreme cases are shown separately to emphasize this point.

From the energy savings graph, it can be seen that *esav* technique performs almost as well as 'esav-best'. It performs about 5.8% better than 'esav-avg' in the average case and about 14% better for the 'extreme' applications. The 0.3% difference between 'esav-pred' and 'esav-avg' gets bridged by using the ED^2 analysis. However, this requires an apriori knowledge of the voltage setting. From the slowdown graph, it can be observed that the slowdown due to these techniques is minimal. That is the reason why the ED^2 analysis doesn't benefit 'esav-pred' much. With very little performance overhead, the optimal ED^2 point lies close to optimal energy point itself.

It is interesting to note that dynamic adaptation schemes do not perform as well as 'esav-pred' or even 'esav-avg'. The reasons for this, we believe, are two fold. One is the difficulty in tuning their parameters and the other is the high level of noise inherent to access interval patterns. The performance factor (PF) of AMC and the setpoint for IMC are such tough-to-tune parameters. 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.

From the slowdown graph, one can observe that 'esav-pred' is better than 'esav-avg' and other adaptive schemes from a performance standpoint also - however, this difference is marginal - 0.5% at the maximum.

Hence, our profile-based scheme is robust in terms of adaptation, performs reasonably well in the average case and is much better for extreme applications.

6 Conclusion and future work

This paper introduced 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 profile data. Decay interval chosen by the analysis of such profile data matches the optimal decay interval with a reasonable degree of accuracy. This makes such profile based schemes better than the dynamic adaptive schemes and almost as good as an omniscient choice of individual-best decay interval. The profile-based technique achieved its adaptation target successfully - it performed slightly better than the average-best case for most benchmarks and much better than the average-best case for the 'extreme' applications.

Future work in this direction could explore many interesting issues. The most important issue is that of cross-training. An understanding and exploration of how well this profile-based technique performs when the input sets are substantially different is an area of possible further work. Also, extension of this work to make the scheme dy-

namically adaptive by combining it with phase detection 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 an aspect that one might look at. Moreover, application of this technique to the instruction cache, lower level caches and other structures within the CPU is also a possible direction for future work.

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*, Apr. 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] 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.
- [10] 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.
- [11] 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.
- [12] K. Scott, K. Skadron, and J. W. Davidson. Low-overhead software dynamic translation. Technical Report CS-2001-18, University of Virginia Department of Computer Science, Oct. 2001.
- [13] 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, Oct. 2002.

- [14] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, Apr. 2003.
- [15] SIA. *International Technology Roadmap for Semiconductors*, 2001.
- [16] 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.
- [17] 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.
- [18] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.
- [19] 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*, Aug. 2003.
- [20] V. Zyuban. Unified architecture level energy-efficiency metric. In *Proceedings of the 2002 Great Lakes VLSI Symposium*, pages 24–29, Apr. 2002.