Profile-Based Adaptation for Cache Decay

KARTHIK SANKARANARAYANAN and KEVIN SKADRON
University of Virginia

"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 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.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.1.4 [**Processor Architectures**]: Parallel Architecture—*Mobile processors*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Adaptation, cache decay, interval, leakage power

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 [SIA 2001] 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 [Kaxiras et al. 2001] is one such technique that shuts down infrequently accessed cache lines to save leakage power. Once a new line is brought into a cache, it typically gets accessed many times due to the locality

Authors' address: K. Sankaranarayanan and K. Skadron, Department of Computer Science, University of Virginia, 151 Engineer's Way, 204 Olsson Hall, Charlottesville, VA, 22901; email: {karthick, skadron}@cs.virginia.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 1544-3566/04/0900-0305 $5.00

of the program. After the last access, it remains in the cache until another 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 identifies these dead times and switches the lines into standby mode by gating off their power supply (V_{dd})—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. Because gating the V_{dd} or ground [Yang et al. 2001] results in loss of cache state, shutting down live lines leads to extra, *induced* misses to the next-level cache, which slows 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 refetching the lines. On the other hand, if the decay interval is too large, very few dead lines will 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 algorithm" analysis to obtain a universal decay interval value of 8K cycles. However, the optimal value of the 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 behavior. 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.

Such an adaptation is much less important for state-preserving techniques like drowsy cache [Flautner et al. 2002], 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 [Li et al. 2004]. The energy and performance advantage of gated-V_{dd} is likely to be even better than that suggested by Li et al., 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}. Hence, this work focuses on decay interval adaptation for gated-V_{dd}.

1.1 Related Work

Cache power optimization by dynamic resizing is a well-researched area and has given rise to many techniques that save dynamic [Albonesi 1999] and static power [Kaxiras et al. 2001]. Kaxiras et al.'s work improves upon a coarser-granularity work [Yang et al. 2001] on instruction caches, which

shuts down groups of I-cache lines. Both these techniques use the gated-V_{dd} technique. Other state-preserving alternatives to gated-V_{dd} like reverse body bias [Kobayashi and Sakurai 1994; Nii et al. 1998] and drowsy caches [Flautner et al. 2002] have been proposed, but may be inferior for L1 caches that are backed by fast, on-chip L2 caches [Li et al. 2004]. 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 behavior has been looked at by Kaxiras et al. [2001], Zhou et al. [2003], and Velusamy et al. [2002]. 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* behavior without decay. AMC adapts the decay interval by making the application's decay behavior track its true nondecay behavior 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 (PF) for AMC). For per-line adaptive decay, it depends on the choice of its decay interval. Furthermore, IMC and AMC also suffer from the difficulty in tuning these parameters according to application behavior. In this paper, we compare our work to each of the above adaptation techniques.

1.2 Terminology

Since this work is based on cache decay, it is useful to first state the common terminology used in decay-based leakage saving policies. The energy saved in decay schemes is 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. 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 because they could come at a cost of increased performance loss or increased energy dissipation due to extra misses to the lower level.

(2) The dynamic energy cost incurred in going to a farther cache due to turning off live lines. This consists of two parts:
   (a) An extra *induced miss* (*im*) resulting on the next access to a live line that has been turned off.
   (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 behavior 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 [Lee et al. 2000] that are just happening ahead of time.)

(3) The energy overhead due to any additional hardware used to implement the decay policies. In case of constant cache decay [Kaxiras et al. 2001], this is due to the counters used to measure the time of last access. In case of the adaptive decay policies IMC [Velusamy et al. 2002] and AMC [Zhou et al. 2003], this additionally includes the energy of keeping the cache tags on for measuring the *im*.

(4) The energy cost of performance degradation. 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, and so on due to pipeline stalls constitutes the recomputation energy.

1.3 Contributions

This paper provides a profile-based technique for decay interval adaptation that chooses the optimal decay interval through offline analysis. We find that most of the variables mentioned above, which determine the energy saved due to decay, can be estimated with a reasonable degree of accuracy through profiling. Furthermore, 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, and so on.

The remainder of the paper is organized as follows: Section 2 presents the motivation behind our work, Section 3 describes our estimation technique, Section 4 describes our evaluation framework, Section 5 discusses the results, and Section 6 concludes the paper.

2. MOTIVATION

Variability. The optimal decay interval for an application depends on the nature of its memory accesses. As application behavior varies, the optimal decay interval also varies. Table I 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" behavior. The first column of the table shows statistics about access intervals. The second column shows statistics about dead times.

Table I. 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 Interval (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 *across* applications is evident in both cases. Further, not only are the means different, so are the standard deviations—that is, 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 behavior, the optimal decay interval also varies. This suggests that there is room for adaptation in cache decay.

Live lines. The penultimate column of Table I 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 these 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 that 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 outlines the profile-based estimation technique used to choose the optimal decay interval. More details can be found in the technical report version of this paper [Sankaranarayanan and Skadron 2004].

3.1 Normalized Leakage Savings

Among the four variables mentioned in Section 1.2, which the energy saved in cache decay depends on, the energy due to overhead hardware has been shown to be negligible [Kaxiras et al. 2001] and hence can be omitted. However, it cannot be omitted for AMC or IMC, since the tag array dissipates a nonnegligible fraction of the total cache power. Moreover, 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 of baseline cache leakage energy. This leads to the following expression for the normalized cache leakage savings (*esav*):

$$esav = tor - l2_ratio \times (im_per_cycle + iwb_per_cycle) - idle_ratio \times perc_slowdown \quad (1)$$

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 Kaxiras et al. [2001]. *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 power simulator [Brooks et al. 2000] 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} =1.3 V at a clock speed of 3 GHz. This ratio turns out to be 10.6%. We assume that this ratio holds good also for leakage power. Furthermore, assuming the rest of the processor is not using any decay policy, and using the average leakage savings value of 75% from Kaxiras et al. [2001], *idle_ratio* can be found to be $(100 - 10.6 \times 0.75)/10.6 = 8.7$. Including the effect of nonideal 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, namely, *tor*, *im_per_cycle*, *iwb_per_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 successive powers of two.

Let us begin with *im_per_cycle*. Induced misses are those accesses to a particular live cache line whose interoccurrence 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 $\sum_{j=i}^{n} a(t_j)$. Let this be denoted by $A(t_i)$. Hence, if t_i is the decay interval we are interested in, then the number of induced misses corresponding to that is given by

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

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 *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 that 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 *mispredictions*. Hence, with a reasonable degree of approximation, *iwb* can be obtained by multiplying this by the misprediction rate. that is,

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

This equation is approximate because it 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.

The next important variable to be estimated is the turn-off ratio *tor*. It is determined by the time spent both by the live and dead cache lines in the standby mode. Assuming a uniform distribution of the dead times and access intervals between t_i and t_{i+1}, the expected value of the time scale—the *midpoint*, lies at the geometric mean of the end points of the range (t_i and t_{i+1}).

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 $t_i\sqrt{2}$. Hence, the average *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 $AT(t_i)$, 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, that is, $t_i(A(t_i) + D(t_i))$. 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. that is,

$$tor(t_i) = \frac{(AT(t_i) + DT(t_i))\sqrt{2} - t_i(A(t_i) + D(t_i))}{n_lines \times n_cycles} \tag{4}$$

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. Hence, its mean will lie at a constant distance from the geometric mean of the range. Thus, the 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 performance degradation. This is because of the induced misses and hence depends on *im_per_cycle*. Actually, it is the product of *im_per_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. Moreover, out-of-order superscalar issue, speculation and nonblocking caches make this effective latency variable. The effective latency also depends on the criticality of the loads that 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_per_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

$$perc_slowdown(t_i) = eff_lat \times im_per_cycle(t_i) \tag{5}$$

Apart from the performance degradation, we expect some difference between the estimated and the actual values for the other variables too. This is because, we do not directly account for eager write-backs in the analytical expressions. Furthermore, as 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. The decay process also changes the default replacement behavior of the cache because lines that are in the standby mode get evicted before those that are active—regardless of the replacement policy. 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)–(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 to 1M cycles) can be generated by profiling. The profile data thus generated are 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.

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 are 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 powered 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 [Bala et al. 2000] to collect profile data in software and then run natively in hardware. Phase detection and adaptation techniques as in Huang et al. [2003] and Sherwood et al. [2002] could be used to select different profiling points, and the dynamic translation infrastructure [e.g., Scott et al. 2003] 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 reused whenever the phase recurs. Everytime a new phase is discovered, the profiling step could be run in software for that particular phase.

4. EVALUATION

The three adaptive techniques against which we compare our technique are Velusamy et al.'s IMC, Zhou et al.'s AMC, and Kaxiras et al.'s per-line adaptive cache decay. AMC tries to track the default program behavior 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 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. Per-line adaptive cache decay uses a per-cache-line adaptive scheme. After a line is decayed, a counter is used to gauge the time of the next access. Depending on when the next miss occurs (sooner or later), the decay interval is increased or decreased.

4.1 Simulation Setup

The profiling described in the previous section is implemented in the SimpleScalar 3.0c simulator toolset [Burger et al. 1996]. 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

Table II. Summary Statistics of the Benchmarks Used to Evaluate the Techniques in this Study

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

All benchmarks use reference inputs.

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.

The microarchitectural configuration we use to evaluate the techniques resembles Alpha 21364 as closely as possible. The core processor has an 80-entry RUU and can issue up to 4 integer and 2 floating-point instructions per cycle. L1 caches are each 64K, 2-way with 64 byte blocks and an access latency of 2 cycles. L2 cache is 4 MB, 8-way with 128 byte blocks and a 12-cycle access latency. The latency to main memory is 225 cycles. We do not model replay traps. Finally, as described in the previous section, we use Wattch 1.02 power simulator [Brooks et al. 2000] 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 II. 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 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 [Sherwood et al. 2002]. Simulation is conducted using SimpleScalar's EIO traces to ensure reproducible results for each benchmark across multiple simulations.

5. RESULTS

5.1 Accuracy of Estimation

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, and so on) are substantial with respect to the choice of a decay interval.

As a candidate, we study a typical benchmark, gcc. We compare the values of the four variables estimated by profiling against the actual values simulated across a range of decay intervals (256 to 1M cycles). 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. We present the summary statistics here and encourage the readers to look at the technical report version of this paper [Sankaranarayanan and Skadron 2004] for more details. We find that the estimated values of *tor* and $im_per_cycle + iwb_per_cycle$ 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 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_per_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 coefficient between the actual and estimated performance degradation is high at 99.5%. Hence, 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, namely, eager write-backs, change in replacement behavior, and elongation of dead times and access intervals during decay do not impact the accuracy of estimation substantially. Furthermore, 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* behavior 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

Table III. Optimal Decay Interval—Actual and Predicted Values (in K cycles)

Integer				Floating Point			
bench	esav-best	esav-pred	esav-self	bench	esav-best	esav-pred	esav-self
mcf	1	2	**1**	art	0.25	**0.25**	**0.25**
gap	2	8	**2**	lucas	1	**1**	**1**
twolf	4	**4**	**4**	swim	1	2	2
vpr	4	8	**4**	applu	2	**2**	**2**
bzip2	4	**4**	8	equake	2	4	**2**
vortex	4	16	8	wupwise	2	8	**2**
gzip	8	**8**	**8**	facerec	2	4	4
perlbmk	8	16	**8**	galgel	4	2	**4**
parser	16	8	8	ammp	4	2	**4**
gcc	16	**16**	**16**	fma3d	4	**4**	**4**
crafty	16	**16**	**16**	mesa	4	2	8
eon	16	**16**	**16**	mgrid	8	4	**8**
				apsi	16	8	4
				sixtrack	16	**16**	**16**

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 *esav* metric. Table III shows the results of the comparison. The *esav-best* column shows the actual optimum and the *esav-pred* column shows the predicted optimum. Values that match are shown in bold face font. It can be seen that *esav-pred* is off the *esav-best* at most by a factor of 4. The average number of power-two steps by which it is off is 0.7. This shows that the profile-based estimation tracks the actual optimum reasonably well. However, it is not quantitatively clear from this data, how much of the difference is due to cross-training as opposed to inaccuracy in estimation. Isolating these will help us understand the accuracy of estimation better. Hence, we 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 the cache decay simulation process. The results for this experiment are shown in the *esav-self* column. It can be seen that the number of mispredictions drops from 15 to 7. Similarly, the average number of steps by which the prediction is off, goes from 0.7 to 0.3. Thus, it can be seen that more than half of the mismatch stems from cross-training.

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 the individual-best decay interval with respect to energy savings. This oracle scheme is called *esav-best*. We also evaluate a nonadaptive scheme corresponding to the original cache decay work, which sets the decay interval to a constant value of 4K cycles. This is the single-best constant decay interval in terms of energy savings. This static technique is called *esav-avg*. Our profile-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*. The various adaptive schemes we evaluate are: IMC, AMC,

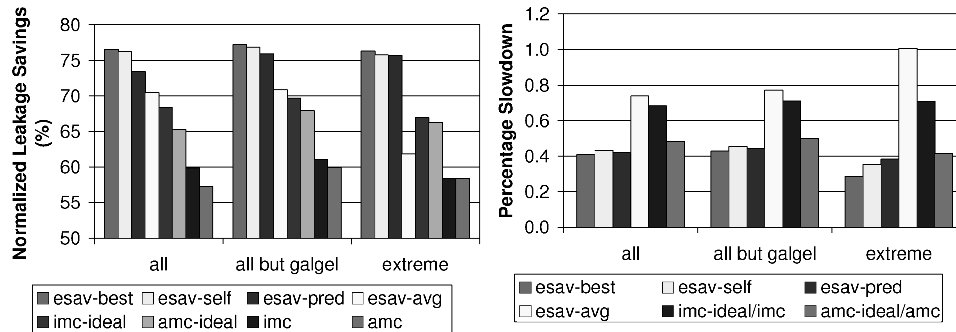

Fig. 1. (a) Normalized leakage energy savings and (b) percentage slow down for the various decay interval adaptation schemes.

and per-line counter-based adaptive decay. The details of the parameters chosen for these techniques can be found in the technical report version of this paper [Sankaranarayanan and Skadron 2004]. 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 1(a) shows the normalized leakage energy savings for the above-mentioned decay schemes in the order of their performance while Figure 1(b) shows the percentage slowdown. In these graphs, the absence of per-line counter-based adaptive cache decay is conspicuous. It is not included because per-line adaptive decay selects overly aggressive decay intervals that 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 behavior is also reported in Zhou et al. [2003]. 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.

Graphs in Figure 1 show 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 behavior, we indicate the data from all benchmarks except galgel separately. The results of the *esav-self* technique also illustrate this behavior clearly. It can be seen that it matches closely with the cross-training (*esav-pred*) results unless galgel is included.

The rightmost group shows averages only among those benchmarks whose optimal decay interval is farther 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 behavior of our profile-based technique. These 10 "extreme" applications are the main motivation behind adaptive techniques like ours. A good adaptive technique should perform at least as well as a nonadaptive 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.

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 nonomniscient 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 1(a) 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. Also, from the slowdown graph in Figure 1(b), it can be observed that the slowdown due to all the above decay techniques is minimal ($<1.2\%$).

The energy savings graph also shows that both the ideal and nonideal 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 suboptimal 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 1 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 PF of AMC and the set-point for IMC are difficult parameters to tune. They vary depending on the application behavior. 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 and so on) filters out the noise through large-scale averaging. Apart from these problems in adaptation, the nonideal 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 nonideal versions perform poorly in case of extreme applications.

It was pointed out in Section 3 that a difference in replacement behavior arises between profiling and actual measurement of decay. Our equations in Section 3 do not take this into account. Hence, in order to explore the impact of this change in replacement behavior on the effectiveness of our profile-based schemes, we ran experiments evaluating the various decay adaptation schemes by changing the associativity of the 2-way cache used in previous evaluations to 1-way and 4-way. The trend in order of performance of the scheme remained the same across different associativities. We do not provide the results here but the details can be found in the technical report version of this paper [Sankaranarayanan and Skadron 2004].

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 robust in terms of adaptation, performs reasonably well in the average case, and is vastly superior for 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 case 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 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 behavior but also the opportunities afforded by profiling in the context of energy efficiency. Similar to Magklis et al. [2003], 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 effective latency. It will not only be useful to the schemes described in this paper but also the profiling community in general. The extension of this study to state-preserving leakage saving techniques is one more area for further exploration. Also, extension of this work to make it dynamically adaptive by combining it with phase detection research is another interesting venue for future research. 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 behavior. 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 like to express our sincere thanks to the editor, the reviewers, and Mircea Stan for their helpful and constructive feedback.

REFERENCES

ALBONESI, D. H. 1999. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. 248–259.

BALA, V., DUESTERWALD, E., AND BANERJIA, S. 2000. Dynamo: A transparent dynamic optimization system. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Languages Design and Implementation*.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*. 83–94.

BURGER, D. C., AUSTIN, T. M., AND BENNETT, S. 1996. *Evaluating Future Microprocessors: The SimpleScalar Tool Set*. Tech. Report TR-1308, University of Wisconsin-Madison Computer Sciences Department. July.

FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., AND MUDGE, T. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*. 147–157.

HUANG, M., RENAU, J., AND TORRELLAS, J. 2003. Positional adaptation of processors: Application to energy reduction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*. 157–168.

KAXIRAS, S., HU, Z., AND MARTONOSI, M. 2001. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*.

KOBAYASHI, T. AND SAKURAI, T. 1994. Self-adjusting threshold-voltage scheme (sats) for low-voltage high-speed operation. In *Proceedings of the IEEE 1994 CICC*. 271–274.

LEE, H., TYSON, G., AND FARRENS, M. 2000. Eager writeback—A technique for improving bandwidth utilization. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*. 11–21.

LI, Y., PARIKH, D., ZHANG, Y., SANKARANARAYANAN, K., SKADRON, K., AND STAN, M. 2004. State-preserving vs. non-state preserving leakage control in caches. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference*.

MAGKLIS, G., SCOTT, M. L., SEMERARO, G., ALBONESI, D. H., AND DROPSHO, S. 2003. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*. 14–27.

NII, K. ET AL. 1998. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*. 293–298.

SANKARANARAYANAN, K. AND SKADRON, K. 2004. *Profile-Based Adaptation for Cache Decay: A Technical Report*. Tech. Rep. CS-2004-14, University of Virginia Department of Computer Science. Apr.

SCOTT, K., KUMAR, N., VELUSAMY, S., CHILDERS, B., DAVIDSON, J. W., AND SOFFA, M. L. 2003. Retargetable and reconfigurable software dynamic translation. In *Proceedings of the International Symposium on Code Generation and Optimization*. 36–47.

SHERWOOD, T., PERELMAN, E., HAMERLY, G., AND CALDER, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*. 45–57.

SIA. 2001. *International Technology Roadmap for Semiconductors*. SIA.

VELUSAMY, S., SANKARANARAYANAN, K., PARIKH, D., ABDELZAHER, T., AND SKADRON, K. 2002. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*.

YANG, S.-H., POWELL, M. D., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2001. 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*.

ZHOU, H., TOBUREN, M., ROTENBERG, E., AND CONTE, T. M. 2003. Adaptive mode control: A static-power-efficient cache design. *ACM Trans. Embedded Comput. Syst. 2*, 3, 347–372.

Received November 2003; revised April 2004; accepted April 2004