Adaptive Cache Decay using Formal Feedback Control

Sivakumar Velusamy, Karthik Sankaranarayanan, Dharmesh Parikh,
Tarek Abdelzaher, Kevin Skadron
Dept. of Computer Science, University of Virginia
Charlottesville, VA 22904

Abstract

This paper argues that adaptive techniques in processor architecture should be designed using formal feedback-control theory. We use the derivation of a controller for cache decay—a technique for leakage-energy savings—to illustrate the process of formal feedback-control design and to show the benefits of feedback control. Control theory provides a powerful framework that simplifies the task of designing an adaptive system. It provides well-known control designs that are easy to tune for performance and stability. Open-loop adaptation, on the other hand, is vulnerable to unacceptable behaviors when presented with workloads or conditions that were not anticipated at design time. In contrast, feedback control responds to unanticipated behaviors and provides robust operation in cases where open-loop designs fail. Since interest in adaptivity only continues to grow, the need for feedback control techniques and formal design techniques can only grow accordingly. Although our work so far leaves several open questions—especially the question of how best to choose the control setpoint—we hope that this paper will demonstrate the value of formal feedback control and provide guidelines for its general application in architecture research.

1 Introduction

Runtime adaptation has become a topic of great interest in the architecture community, because it allows the microarchitecture to adjust according to a particular application's needs. The greatest amount of work has been in power-aware computing, where a variety of techniques have recently been proposed to recognize idle resources or reduced bandwidth requirements and selectively deactivate portions of the system.

This paper presents early work to demonstrate the benefits of *formal feedback control theory* as a design methodology for adaptive techniques, and to develop general guidelines for the application of control theory in computer architecture.

Most adaptive schemes in the architecture literature use *open loop* response. "Open loop" refers to the fact that the magnitude of response is fixed and cannot be adjusted at runtime to ensure that the desired behavior is obtained. Open-loop techniques work well when the application's behavior is known beforehand, but open-loop techniques can perform poorly in unpredictable systems and for unanticipated workloads. *Feedback control* closes the loop by defining target metrics and error terms for the system being controlled, monitoring the error, and guiding adaptation to minimize this error. Feedback control can therefore allow the adaptive response to adjust to a wide range of behaviors and can respond to unanticipated workloads or behavior.

Feedback control can be designed without using formal control theory, leading to arbitrary control loops and empirically determined control parameters. In many cases these systems work perfectly well. Yet ad-hoc feedback systems are difficult to tune (often requiring exhaustive parameter-space searches), are prone to instability, and are difficult to analyze formally. *Formal* feedback control theory uses difference or differential equations as a fundamental analysis tool. It is often no more difficult—and often easier—to take a formal approach to controller design. Control theory as proven successful in the engineering community in controlling a vast variety of nonlinear, stochastic, and time-varying physical systems. The popularity and success of control theory is in part due to the wealth of developed results that provide well-known controller designs that are easy to tune and analyze, allow designers to reason about the behavior of their control loops and prove bounds on their behavior,[1] and allow designers to use closed-loop properties to force that behavior to adhere to a set of performance and stability specifications using a well-understood analytic approach. An equally important factor to the success of control theory lies in its robustness in the face of modeling errors and external disturbances. The performance guarantees obtained from control-theoretical analysis remain largely unaffected even when the controlled system has not been accurately modeled in the analysis, or when external factors that have not been accounted for at design time interfere with its operation.

Related Work. The design of control systems is of course a mature field with a history dating back at least as far as the 1600s. Numerous textbooks exist that describe its basic principles, *e.g.* [5, 6]. Control-theoretic approaches have been applied to a variety of aspects of computer systems design outside the computer-architecture realm, including CPU scheduling [14, 24], web server quality-of-service management [13, 15], and Internet congestion control [8]. At the circuit level, voltage scaling [4] and current-canceling for leakage control [25] use feedback control, but without using control theory to derive stable control or to prove properties about the resulting system. The only use of *formal* feedback control that we are aware of in the architecture literature is our own prior work on temperature regulation [19].

Contributions. This paper provides general guidelines on how to apply formal controller-design techniques to a desired adaptive behavior, including how to model the underlying system behavior, how to select a setpoint (target metric), how to select sampling rates, and how to implement controllers in hardware. This is most easily demonstrated using a concrete example, for which we choose *cache decay*.

Formal feedback control is not a panacea. It can be difficult to find the correct runtime metric to control, difficult to find a suitable controller design, and difficult to tune the controller to provide acceptable performance. However, the formal part of

[1] Other ways of proving bounds are of course possible, *e.g.* the competitive-algorithms approach in [10].

feedback control is usually not the problem, but rather understanding the system and the right way to abstract it into a control loop. These problems are present regardless of whether any formal control techniques are even being considered. Although the formalism of control theory can be intimidating, we have always found that it *helps* to guide our design and understanding of the system, and found that once we discover suitable control mechanisms, the formal analysis is straightforward and usually simplifies tuning of the final design.

Despite the occasional challenges of formulating feedback control, we argue that with the increasing use of adaptive mechanisms, *the need for closed-loop response to ensure robust behavior is inevitable.* We propose that the design of these feedback control systems should start from formal, control-theoretic formulations. These formulations are no harder to work with than ad-hoc control loops, are often able to simplify the design and tuning process, and make the system easier to analyze.

2 Cache Decay

Cache decay [10]—the running example throughout this paper—is a technique for leakage-energy savings that waits for some pre-determined time, the *decay interval*, before concluding that a cache line's data is no longer in use and that the line can be deactivated. In almost all formulations of cache decay, regardless of whether cache decay is performed using gated-V_{dd} or lower-overhead techniques like reverse body bias [11, 17], the decay interval is a fixed parameter that is determined empirically at design time, thus creating open-loop response. For example, an application that strides through some portion of the cache at a rate just slower than the decay interval will miss on every cache line even if the data fits in the cache. Even if the decay interval is reasonable for all anticipated applications, this average setting may be substantially sub-optimal compared to the ideal value for each individual application, as we show for *perl* in Section 5.

Feedback Control for Cache Decay. Zhou *et al.* [27] observed this variable application behavior for cache decay, and proposed an informal feedback-control cache-decay scheme that they call Adaptive Mode Control (AMC). Unlike open-loop cache decay, which decays both the tag and data portions of a cache line, AMC keeps the tags powered on to track the rate of *induced misses* (misses caused by mistakenly deactivating lines that contain data still in use) relative to the true miss rate. AMC doubles or halves the decay interval if the induced-miss ratio exceeds a specified threshold. Unfortunately, the AMC work did not go on to directly evaluate whether the proposed feedback-control scheme actually adapts to changing application behavior, nor did they evaluate the hardware cost for implementing feedback control.

This paper shows how to develop a formal feedback-control design for cache decay that uses the same tag-based technique for identifying and controlling induced misses. We show that both the formal and AMC feedback designs do adapt to changing application behavior, shows that they both guard against divergent behavior that makes open-loop decay behave badly, and evaluates the hardware cost. The formal controller, which we call IMC (integral miss control), gives almost the same performance as AMC, and IMC was easy to design and tune. These observations suggest that the ad-hoc approach confers no advantage, and so there is no reason not to use the control-theoretic approach when designing closed-loop response.

Cache-Decay Background. Cache decay using gated-V_{dd}, proposed by Kaxiras *et al.* [10], builds on prior, coarser-granularity work by Yang *et al.* [26]. The basic idea is to use counters to detect when a line in the cache has been idle for some time and from this, infer that the data stored there is *decayed*—no longer in use. Gated-V_{dd} saves leakage by disconnecting idle lines from the supply voltage when a line's idle counter reaches its maximum. This loses state within the cache line and requires the valid bit to be turned off. This is harmless if the next access to that line would have been an eviction; but if useful data was discarded, the next access will be an *induced miss.*

Kaxiras *et al.* use a global counter that counts from zero up to one-fourth the decay interval and then starts over. Each line uses a local two-bit counter; when the global counter reaches its maximum value, all two bit counters are incremented. When a two-bit counter reaches its maximum, the line has been idle for the full decay interval; it is assumed that the line's usefulness has decayed, and the line is deactivated. The AMC decay mechanism also uses a global counter to track elapsed time and local, per-line counters to track per-line idleness. The details differ slightly from the technique proposed by Kaxiras *et al.*, but the essence of its operation is the same.

AMC [27] and the per-line adaptive scheme from the original cache decay paper [10] are the only cache-decay mechanisms that we know of using feedback control. Like AMC, the per-line scheme from [10] adapts by multiplying or dividing by two. Instead of storing the per-line decay interval or doing the arithmetic within each line, per-line decay uses a set of global decay counters and each line chooses the decay interval that it wants: if the per-line counter is at either extreme, the interval selects the twice-larger or half-smaller interval. Unfortunately, this requires a somewhat expensive multiplexor per line. The dependence upon the local counters also means that the sensitivity of this scheme to noise cannot be tuned for performance or stability.

3 Evaluation Framework

3.1 Performance, Power, and Temperature Simulation

Performance and Miss/Decay Rates. To model cache decay, we extend the cache-decay simulator that Kaxiras *et al.* used in their work [10]. Our extensions consist of supporting associativity for cache decay, adding the AMC technique, and of course adding our controllers. For our work here, we approximately model the Alpha 21264, as shown in Table 1. The hybrid branch predictor [16] is SimpleScalar's slightly simplified version, using bimodal predictors as the chooser and as one of the components. We do not model the register-cluster aspect of the 21264.

AMC Modeling and Validation. AMC proposed the idea that we have adopted, that the cache tags are never put to sleep. This allows the separation of overall misses into ideal misses and induced misses. AMC achieves adaptivity by doubling or halving the decay interval in response to induced misses. This heuristic operates as follows: using a sampling interval of one million cycles, the number of induced misses is compared to the number of ideal misses. If the number of induced misses is within a specified tolerance, the decay interval in left unchanged; otherwise it is doubled or halved in the appropriate direction. The tolerance, or *performance factor* (PF), is specified as a fraction–50% in the AMC work. In other words, the observed miss rate can vary within $\pm 50\%$ of the ideal miss rate. Large PFs yield very high turn-off ratios and for high miss rates, this results in poor energy savings for a number of benchmarks. We selected a PF for our experiments by testing a range of PFs and choosing the value that gives the best overall energy sav-

Processor Core	
Instruction Window	80-RUU, 40-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 1-cycle latency
L2	Unified, 2 MB, 4-way LRU, 32B blocks, 11-cycle latency, WB
Memory	100 cycles
Branch Predictor	
Branch predictor	Hybrid: 4K bimod and 4K/12-bit/GAg 4K bimod-style chooser
Branch target buffer	1 K-entry, 2-way

Table 1: Configuration of simulated processor microarchitecture.

ings. Because energy is a function of execution time, this same PF also gave good execution performance, with small degradations in IPC. (For zero degradation, of course, the PF can simply be set to zero, disabling decay.)

We replicated as exactly as possible the AMC technique. We validated our implementation by using the processor configuration specified in the AMC work and compared results for our four SPEC95 benchmarks to those reported in the AMC work. All of the data cache configurations in that paper were simulated. The IPC degradation and the *turn-off ratios* (mean ratio of deactivated to active lines) match very closely with results from [27].

Leakage. Some other work on leakage energy, like [7, 10], has already explored the sensitivity of cache-decay techniques to various technology parameters. This paper instead focuses on design and performance issues for feedback control, so we take a single set of energy parameters. The behaviors we observe and the guidelines we present are not closely tied to any particular set of assumptions about energy modeling.

For cache decay, we are primarily interested in modeling leakage energy. As in [10], we use data from Yang *et al.* [26]. For leakage power, they have estimated that with a 110°C operating temperature, a 1GHz operating frequency, a supply voltage of 1.0V and a threshold voltage of 0.2V, an SRAM cell consumes about $1740 * 10^{-9}$ nJ leakage energy per cycle. This is roughly in line with industry data we obtained from Agere's COM3 and COM4 process generations [9]. Based on this data, we computed that each line in the cache leaks $8.9 * 10^{-4}$ nJ per cycle. The operating temperature of 110°C is a reasonable expectation for a next-generation process generation according to the SIA roadmap [18].

When applying gated-V_{dd}, typically only about half this leakage energy is saved. This is because, when the gate transistor is between V_{dd} and the cell, as proposed in [26], there are still some paths to ground for leakage. The most significant is leakage from the precharged bit lines to ground. Gated-V_{ss} is much more effective, because it puts one more off transistor on the path to ground, reducing leakage by about 90–95%. Gated-V_{ss} however has a higher cost to reactivate a line. This is due to charge accumulation. Agere data suggests that a reasonable estimate for the per-line reactivation cost for gated-V_{ss} is 1.89 nJ [9]. Previous cache-decay studies have not accounted for either of these effects (the need to use gated-V_{ss} rather than gated-V_{dd} and the corresponding reactivation cost).

To compute total energy savings—which we refer to as energy *profit* or E_p—we first compute the leakage energy saved using the per-line leakage numbers detailed above, the number of lines decayed, and the number of cycles the lines stay decayed. In doing so, we assume that it takes 5 cycles for a line to be deactivated or reactivated [9]. To account for the costs of decay, we subtract the energy overheads caused by the induced misses and induced writebacks due to the extra accesses to the lower level (L2 in our case). The energy estimate we use for an L2 access is 8 nJ. For any line that is reactivated, regardless of whether it is for an induced or an ideal miss, we charge the reactivation cost. Finally, we factor in the effect of performance loss due to induced misses and induced misses in our energy calculations. We assume an average CPU dynamic energy dissipation of 31.8 nJ per-cycle (This number was obtained by running various benchmarks on the Wattch simulator [3] for a 0.18 μ process, 2.0V V_{dd}, and 1 GHz clock rate) and multiply this by the performance degradation (in number of CPU cycles) due to decaying cache lines. We then subtract this from the energy value above to obtain the total energy savings or the profit that takes into account both the energy saved and the energy overhead due to induced misses.

Why Gated-V_{dd}? Since the original cache-decay work, Hanson *et al.* [7] have explored techniques for saving leakage energy in the cache without losing state, and many other research groups are searching for even better techniques. Hanson *et al.* examine dual-V_t and reverse-body-bias (RBB) designs (which they call MTCMOS). Dual-V_t is a static design that imposes a high-threshold (that is, high V_t) transistor on the periphery of the cache circuitry, reducing leakage from the low-V_t transistors at the cost of slower read times. RBB [11, 17] controls the backgate bias to raise the effective V_t. Like gated-V_{dd}, RBB does not increase the cost of cache accesses for active lines, and RBB has the added advantage that deactivated lines do not lose state. But deactivated lines take some time to be reactivated, which is similar in some ways to an L2 cache miss. RBB also has a much higher cost to activate and deactivate lines. Both effects mean that, just like with gated-V_{dd}, deactivation must be performed judiciously using a decay interval.

For this paper, we use the original, gated-V_{dd} instead of some other technique like RBB for two reasons. First, we wish to compare our formal controller against the AMC technique proposed by Zhou *et al.*, who studied the original decay mechanism. Second, there is a greater range of literature available, which we found helpful in validating our decay and energy models.

It is important to note that the focus of this paper is on the versatility of control theory. Our work is not intimately tied to the use of gated-V_{dd}. The controllers and analysis we present are common formulations that are straightforward to map to other adaptive mechanisms within a computer system that have nothing to do with caches—for example, resizing of the instruction queue [1] or choosing a voltage/frequency scaling level [4]. Mapping these formulations to an alternative cache-decay mechanism like RBB is even easier, because the same dynamic model can be used and only a few parameters need be changed.

3.2 Benchmarks

We evaluate our results using a mixture of benchmarks from the SPECcpu95 [23] and SPEcpu2000 [22] suites. Summary statistics are given in Table 2. Because our department has a mixture of big-endian and little-endian machines, we used a mixture of benchmarks compiled for the Alpha and PISA instruction sets. The benchmarks are compiled and statically linked using either the Compaq Alpha compiler (with *peak* settings) or SimpleScalar gcc-PISA compiler (with *-O3 -funroll-loops*) and

	FF	Miss rate (Dcache)	Miss rate (Icache)	IPC	T_d (best)
spec95					
gcc	221 M	1.46%	0.23%	1.82	32 K
go	926 M	0.47%	0.05%	1.76	64 K
ijpeg	824 M	0.36%	0.00%	2.31	16 K
li	271 M	0.42%	0.00%	1.94	64 K
spec2k					
crafty	2 B	1.23%	0.02%	2.36	32 K
parser	2 B	2.15%	0.00%	1.69	32 K
perl	2 B	0.43%	0.16%	1.89	512 K
vpr	2 B	4.38%	0.00%	1.78	8 K

Table 2: Summary statistics used for the benchmarks in this study. All benchmarks use reference inputs. Note that the best fixed decay interval is probably not actually a power of two.

include all linked libraries. A given program was always run on the same system type and environment to preserve reproducibility. For each program, we skip some fast-forward interval to avoid unrepresentative startup behavior at the beginning of the program's execution, and then simulate 500 million (committed) instructions using the reference input set. SPEC95 fast-forward intervals were taken from [20] and SPEC00 fast-forward intervals were set at two billion committed instructions, which seems to be a common value.

The benchmarks in the AMC work used only SPEC training inputs which ran for a very short time. The use of reference inputs and longer simulation times gives the controllers more opportunity to demonstrate their adaptability—or to demonstrate oscillating or saturating behavior.

4 A Roadmap for Applying Control Theory in Computer Architecture

In this section we give a framework for designing feedback control by describing the process for designing a simple but effective digital *integral* controller for cache decay. In the rest of the paper, we evaluate this controller against open-loop and AMC cache decay. Because, to the best of our knowledge, the formalities of digital feedback control have not been presented elsewhere in the computer-architecture literature, this section presents these formalities in some detail for instructive and easy reference purposes. Although the formalism of control theory can be intimidating, in practice we have found that its use is in fact easily learned and usually amenable to a straightforward, "cookbook" approach. Furthermore, we have always found that the formalism helps to guide our design and understanding of the system, forcing us to attend to potential problems that we would otherwise have only found much later.

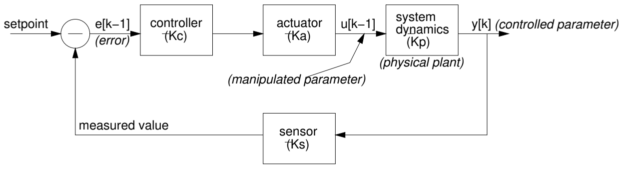

Figure 1: A typical feedback control system.

4.1 Mapping Adaptive Techniques onto Control Loops

The Control Loop. A typical feedback control system, shown in Figure 1, is composed of a control loop with a setpoint, a controller, a system or "physical plant" being controlled, an actuator which manipulates some parameter within the system, and a sensor. The *setpoint* of the control loop is the target value of the *controlled parameter*, which for example might be a specified number of induced misses. The controller output is computed from the *error* between the setpoint and the value measured by the sensor for the controlled parameter. This output is translated by the actuator into some change in the manipulated parameter—a change in the decay interval T_d in our example. This change then affects behavior of the system (more or fewer lines are deactivated) and hence changes the behavior of the controlled parameter (more or fewer induced misses are observed).

In many systems, the actuator and sensor do not disturb the signal, so they can be omitted from the control loop. That is the case here. Typically, the sensor and actuator are only relevant if they scale the input, in which case their gain is equal to their scaling factor. Note also that the units going around the loop must be consistent. This is essential for analysis and an accurate computation of K_c. It is also an excellent sanity check that has uncovered many flaws in our early controller designs.

For straightforward design, the setpoint must be a value that can be tracked, rather than an unquantifiable minimization or maximization. The manipulated variable is the parameter that is implementing the adaptation; for cache decay it is the decay interval T_d. The choice of setpoint dictates the controlled variable. For cache decay we chose the setpoint in terms of a reference rate of induced misses.[2] The controller therefore attempts to keep the induced miss rate as close to this setpoint as possible. Note that the setpoint is not zero. This would drive a feedback system to values of T_d high enough to stop deactivation altogether. We chose a rate because it yields the simplest control loop. Note that the AMC target that is based on a rate with a performance factor does not easily translate to a setpoint. It yields a non-linear controller, as seen in Figure 2, that is more difficult to analyze. More discussion of these two different targets for the controlled parameter appears in Section 5.

For our implementation, we chose the value for the setpoint by first determining the best constant (non-adaptive) decay interval data for various benchmarks. Then the induced miss rate that corresponds to the best average performance in terms of IPC and energy savings was selected as the setpoint. Closed-loop control then adapts to varying application behavior by holding the induced-miss rate to the setpoint.

Continuous vs. Digital Control. Linear control techniques can generally be classified into continuous or analog control and digital or discrete control. The two techniques differ in the way they model the underlying system for the purposes of controller design. Analog control assumes that the controlled system produces a continuous output described by a differential equation. Examples of such systems include physical processes operating in continuous time. Digital control assumes that the output of the controlled system is relevant (or known) only at discrete (usually equidistant) points in time. The output is therefore described by a sequence of numbers generated by a difference equation. Sampled systems, like the one we develop here, are often described by discrete time models.

[2] For units agreement, the control loop requires the setpoint to be in terms of number of misses rather than a rate. Since we used a fixed sampling rate, this is straightforward.

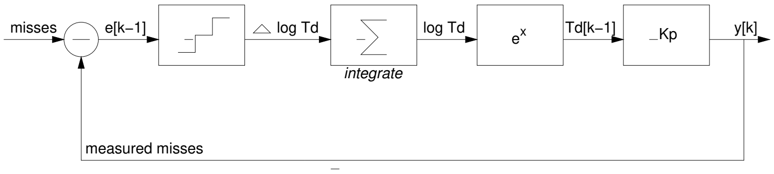

Figure 2: The AMC control loop.

Proportional Response. The most common controller design is proportional response, which for digital control can be expressed by:

$$u[k] = u_0 + K_c e[k] \tag{1}$$

Where $u[k]$ is the value of the manipulated parameter (decay interval for cache decay) for sampling period k, u_0 is some fixed offset determined off-line, K_c is the *controller gain*, and $e[k]$ is the error observed at sampling period k. The gain of this system is K_c. The relationship between the manipulated variable $u[k]$ and the controlled variable $y[k]$ is modeled by a *process gain* K_p such that we expect:

$$y[k] = K_p u[k-1] \tag{2}$$

(For linear control theory, K_p is a constant. Many non-linear systems can be linearized, but that is beyond the scope of this paper.)

Proportional response works well when the system is expected to fluctuate about a steady-state value u_0 for the manipulated variable. Because integral response is more flexible and the analysis is more interesting, we will not further consider proportional control.

More About Process Gain. K_p is an inherent property of the physical plant that conveys the magnitude of the system's response to a change in the manipulated parameter—in our example, K_p conveys how a change in T_d affects the *expected* induced-miss rate. 5observed at sampling instant k, is determined

The purpose of control-theoretic analysis is to choose K_c to match K_p and avoid unstable behavior. If K_c is chosen without consideration for the system's magnitude of response and K_c is too large, then a sufficiently large error (due to unanticipated system behavior or noise) can cause the system to become unstable and oscillate: a large error causes a large change in $u[k]$, which causes a large error in the other direction and so on. Even if oscillation does not ensue, the large gain often means that the controller has difficulty tracking the setpoint and inferior performance results. One of the major benefits of this simple control theoretic analysis is therefore not just the guarantee of stability, *but the formal determination of controller gain to efficiently track the setpoint.* With adequate dynamic models of the system, control theoretic analysis therefore provides the correct value of gain "for free"—a major motivation to use formal rather than ad-hoc feedback control. We consistently found that tuning the gain empirically was extraordinarily difficult, requiring large parameter-space searches, but with control theory and an adequate model of the system's dynamics, the correct gain simply "falls out" of the analysis.

Deriving K_p is the subject of Section 4.2.

Integral Response. A more flexible and responsive controller that is better suited for cache decay is integral response, which for digital control can be expressed by:

$$u[k] = u[k-1] + K_c e[k-1] \tag{3}$$

The only change from equation 1 is that the value of the manipulated parameter remembers all the previous changes: $u[k] = u[0] + K_c e[1] + K_c e[2] + ... + K_c e[k]$ For cache decay, equation 3 becomes:

$$T_d[k] = T_d[k-1] + K_c e[k-1] \tag{4}$$

where T_d is the decay interval. This is an intuitive way to make cache decay adaptive. Regardless of the initial state of the system, T_d will converge to track the system behavior. Once a value of T_d is found that eliminates the error, T_d will become constant until the system behavior again changes. This controller is pictured in Figure 3.

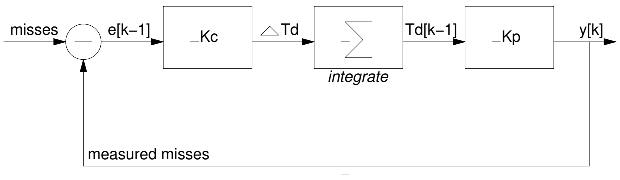

Figure 3: The control loop for our revised cache-decay controller in equation 4

Next we take the z-transform of Equations (2) and (3) to compute a *transfer function* for the individual blocks and the loop as a whole in Figure 3. The transfer function simply gives the relationship between the input and output signals and are typically expressed using the z-transform. The z-transform is a widely used technique in digital control literature that transforms difference equations into equivalent algebraic equations that are easier to manipulate. It is the digital equivalent of the Laplace transform for frequency-domain analysis. While a review of z-transform is outside the scope of this paper, the two main properties that make it extremely useful for difference equation analysis are presented for completeness below:

- *Property 1:* If the z-transform of a sampled function $x[k]$ in the discrete time domain is denoted by $X(z)$, the time-shifted function $x[k-n]$ has the transform $z^{-n}X(z)$. Translation of time-shifts into powers of z allows translating difference equations into algebraic ones by means of z-transform.
- *Property 2:* If f_1 and f_2 are two functions in the discrete time domain, the z-transform of $f_1(f_2(k))$ is $F_1(z)F_2(z)$, where $F_1(z)$ and $F_2(z)$ are the z-transforms of f_1 and f_2. This property allows the reduction of large signal flow diagrams of cascaded blocks into the product of the block's respective z-transforms.

From the above rules, it is easy to see that the z-transform of Equations (2) and (3) lead to the relations:

$$\frac{y(z)}{u(z)} = z^{-1}K_p \tag{5}$$

$$\frac{u(z)}{e(z)} = \frac{z^{-1}K_c}{1-z^{-1}} \tag{6}$$

giving us the transfer functions for the two major blocks in Figure 3. That loop can now be redrawn to show these transfer functions, as in Figure 4.

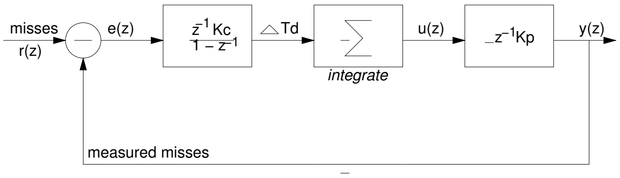

Figure 4: The control loop from Figure 3 showing the transfer equations for the control and process blocks.

Selecting the Controller Gain and Stability Analysis. To select the gain K_c, we wish to choose a value that is commensurate with the behavior of the system to rapidly track changes while giving stable behavior. Stability analysis meets both of these goals by relating the K_c to the impact of any other transformations in the control loop. From the product of Equation (5) and Equation (6), and because $e(z) = r(z) - y(z)$, we follow the loop in Figure 4 to obtain:

$$y(z) = \frac{K_c K_p z^{-2}}{1 - z^{-1}} \left(r(z) - y(z)\right) \tag{7}$$

$$\frac{y(z)}{r(z)} = \frac{\frac{K_c K_p z^{-2}}{1-z^{-1}}}{1 + \frac{K_c K_p z^{-2}}{1-z^{-1}}} \tag{8}$$

This gives the transfer function for the loop as a whole and puts the equation in the form needed for stability analysis. The significance of the above expression lies in that it relates the set point or design objective r, to the actual value of the controlled variable y. If y were to track r perfectly at all times, the above transfer function would have been equal to unity. This, however, is not realizable due to system delays, inertia, and similar factors. Instead, stability analysis defines the conditions under which y asymptotically approaches r. Stability analysis begins by computing the values of z that set the denominator of Equation (8) equal to zero, i.e, solving the equation:

$$z^2 - z + K_c K_p = 0 \tag{9}$$

This is known as the *characteristic equation* and in this case gives us a quadratic formula. Because of the square root in the quadratic formula $-b \pm \frac{\sqrt{b^2-4ac}}{2a}$, the roots lie somewhere in the complex plane. Stability requires that the roots of the characteristic equation be within a unit circle from the origin of the complex number plane. Rather than solving this manually, once the characteristic equation (equation 9) is known, *Jury's test* [12] is a simple way to derive the stability constraint. For quadratic equations $f(z) = a_2 z^2 + a_1 z + a_0$, with $a_2 > 0$, no root will be on or outside the unit circle provided that:

$$f(1) > 0 \tag{10}$$

$$f(-1) > 0 \tag{11}$$

$$\mid a_0 \mid < a_2 \tag{12}$$

For equation 9, (10) requires that $K_c K_p > 0$; (11) that $K_c K_p > -2$, and (12) requires that $K_c K_p < 1$.

4.2 Establishing Dynamic Models

Determining K_p, K_a, and K_s requires a model for the dynamics of the plant, actuator, and sensor. For cache-decay mechanisms, the controller computes T_d directly. The actuator simply enforces T_d and hence does not contribute to loop dynamics. For simplicity, we also assume an exact sensor that uses the cache tags to distinguish induced misses from genuine misses, so $K_s = 1$.

We also assume for simplicity that K_p is a scalar, which is required for the linear analysis given above and implies that the system response in terms of induced misses is linear with respect to the decay interval. In fact, as Figure 5 shows, the response is exponential (note the logarithmic x-axis). Yet K_p need only be an approximate model of the system. Inaccuracies in the controller's runtime estimation of T_d will be corrected by feedback. We want to design a conservative controller that will be stable regardless of how strongly T_d affects the number of induced misses; as long as we design for a worst-case value of K_p that represents the system's strongest response to a change in T_d, then when we have less than worst case behavior, the controller will adapt somewhat more slowly.

For the control loop in Figure 3, K_p must be in units of lines/cycle. The input to the physical plant is T_d in units of clock cycles, and the output is the number of induced misses in terms of cache lines. We use units of "lines" to avoid confusion between induced and genuine misses; a cache line can have at most one induced miss per decay interval, because lines can only be deactivated at the end of each decay interval.

Some systems have an inherent or natural response that requires no further modeling. For example, the thermal work in [19] takes advantage of the thermodynamic fact that the steady-state response of temperature as a function of power dissipation is given by the thermal resistance. K_p is therefore simply the thermal resistance, and this step is trivial.

More generally, the system model may be computed dynamically from input-output measurements. Such measurements must be taken over a sufficiently large interval of time to average out noise. When the input is varied randomly, the resulting change in the output (controlled variable) gives the system response. Least-squares estimation is commonly used to fit input-output measurements to a difference equation. This is called model estimation. The resulting equation can be used for the control analysis demonstrated above (in place of Equation (2)).

A difference equation model inherently assumes that the current value of the controlled variable is correlated to a finite sequence of past inputs. If the system has no "memory", the current value of the controlled variable is unlikely to be correlated with old inputs. Hence, the difference equation model collapses to a single coefficient that relates the *current* input to the *current* output (without consideration to the past). In many cases, a sufficient way to estimate system response is to measure the slope of the curve relating the steady-state or average values of the controlled variable to the corresponding values of the manipulated (input) variable. The advantage of using average response is that it cancels out noise and prevents it from giving erroneous values for K_p. Measuring average response is easy to do for cache decay. We simply run, for a representative sample of benchmarks, simulations in which we measure the average rate of induced misses for a given T_d. If we repeat this for several values of T_d, we obtain curves like the one for SPEC95 *perl* in Figure 5. *Perl* is the benchmark that gives the steepest slope, although the slope for several other benchmarks is nearly as steep.

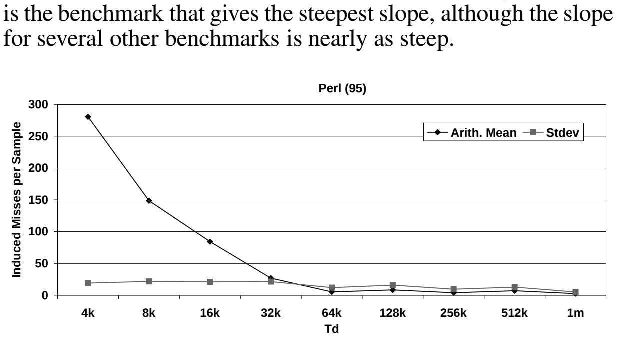

Figure 5: Average number of induced misses as a function of decay interval for SPEC95 perl.

The worst-case K_p (the steepest slope) here is -0.033. Our stability criterion requires that $K_c K_p < 1$, or $K_c = 1/K_p$. So $K_c \simeq 30$. This is the threshold for stability, so we reduce K_c by a *damping factor* of 4 (a typical value) to choose a gain of 8.

We did test the effect of different gains; in practice, we find that the system's performance is insensitive to gain in the range 4–32, but above 64 we begin to get more energy savings for some benchmarks (the controller can change T_d faster in response to a change in program behavior), but oscillation for other benchmarks.

Often the best method to determine system response is obvious, as with the cache decay and thermal examples. If not, it may be necessary to try measuring both impulse and average response to determine which is better.

4.3 Sampling Rate

For digital control, the final aspect of designing the controller is the choice of sampling rate. For some systems, the sampling rate may be dictated by the system or by physical limitations of the sensor. In our case, the decay interval dictates a lower bound on sampling rate, because we do not wish to change T_d in the midst of a counting interval and hence the fastest rate we can use corresponds to the largest T_d we consider. For two-bit line counters and a max T_d of 512 Kcycles, the minimum sampling rate is 128 Kcycles.

Our cache-decay system accumulates a count of induced misses over the last T_d. If samples are an accumulated measurement like this, slower sampling rates average out bursty behavior but slow the controller's response, creating a tradeoff. The sampling rate needs to be fast enough to respond to important changes in the program's behavior. For example, for programs with phase behavior, like the time-domain plot for *compress* in Figure 6, we want T_d to adapt early in each phase. Otherwise T_d lags significantly and its value is often a mismatch for the program's actual behavior. *Ijpeg* has an even more prominent period of about 2 million cycles. *Gcc* in contrast has no obvious period in the time-domain data, and other benchmarks fall between these examples.

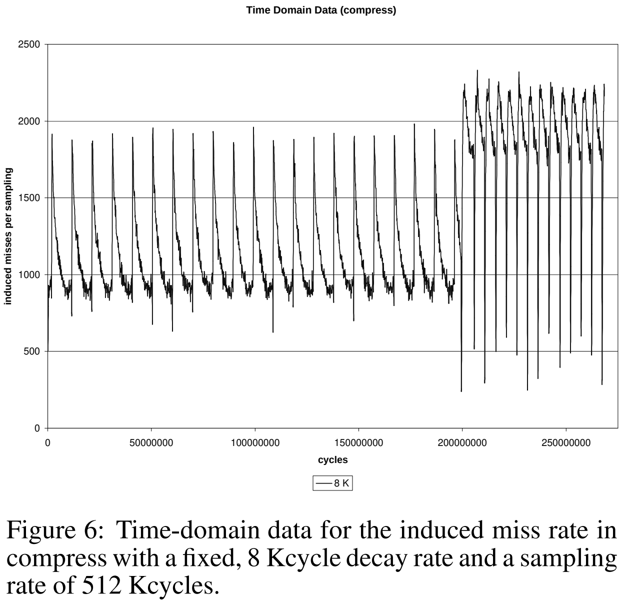

Figure 6: Time-domain data for the induced miss rate in compress with a fixed, 8 Kcycle decay rate and a sampling rate of 512 Kcycles.

We set our T_d at 512 Kcycles. This is slow enough to average out some noise (because the maximum rate is every 128 Kcycles, we are effectively taking a boxcar average of each 4 samples), but fast enough to respond promptly even to the short phases of *ijpeg*.

We also found that clipping was essential due to occasional, high-amplitude noise (for example, when the working set is changing but the underlying decay rate that corresponds to the program's usage pattern does not change). We settled on a clipping value of +/- 50% of the setpoint.

4.4 How to Build Practical Systems

The only remaining question is how to implement formal feedback control in a practical way within a microprocessor—the control computation from equation 4 must be computed every sampling interval. The main issue is the potential cost in terms of execution time (if done in software) or area and power (if done in hardware). Although the above derivation may have made it appear that the resulting controller is complex, in fact many control loops are so simple that they consist only of a few adds and multiplies. For example, assembly code for our sample cache decay controller appears in Figure 7.

Controller Implementation. The simplest solution is a purely hardware implementation using dedicated arithmetic units. Although the integer execution unit in most microprocessors is large, this is because these units are part of a large and complex datapath, and designers typically maximize the performance to maximize clock rate. This leads to transistor sizings of 10 X or more [21]. Dedicated hardware need not drive large capacitive loads and need not compute so fast, allowing close to minimum sizing. This makes the area and power of the requisite controller hardware negligible.

The hardware requirements for our controller and for AMC are fairly similar. For cache decay, note that any feedback system requires registers for the current value of the controlled variable (T_d), and the current value of the observed variable $y[k]$ (the number of induced misses). The implementation of our integral controller only requires the addition of an adder, a multiplier, a comparator, a shifter, and a PLA or ROM to drive them (note that for low-speed design, neither PLA nor ROM will have significant leakage). The size of the PLA/ROM is proportional to the size of the state machine needed to drive the computation in Figure 7. Typically, these can be fixed-point units with modest precision; for our controller, integer units suffice.

Other Costs. The control computation itself can be implemented at negligible area and power cost. But other aspects of the controller design may present significant expense, and these must be accounted for as well.

Cache decay presents an example. Both our controller design and the AMC design require that the tags not be decayed, so that induced misses can be distinguished. Since the leakage in the tags is about 3.5% of the entire leakage of the cache (for 32-bit addresses and 64-byte lines), this presents a substantial lost opportunity. And as we see in the next section, this expense is so large that it causes both feedback-control techniques to give lower energy profit for some benchmarks than does simple open-loop decay.

5 Evaluating Cache-Decay Control

In this section, we evaluate the integral controller (IMC) described in the previous sections, comparing it to AMC and to a simple open-loop controller that uses a single fixed T_d of 32 Kcycles. On the assumptions that these parameters must be set at the processor's design time and that in any case profile-guided feedback is unlikely, we have chosen tuning parameters for all

```
IM  <- IM - 472           ; compute error
IM  <- IM & 0xff          ; clip error to +/- 255
IM  <- IM << 3            ; apply controller gain by computing Kc * error
TD  <- TD + IM            ; Td[k] = Td[k-1] + Kc * error
TD  <- TD & 512k          ; clip Td at upper bound = 512k
tmp <- Td & 0xfffff000    ; test whether new value of Td < 4k
if (tmp == 0)
   then Td <- 4k          ; if so, clip Td to lower bound = 4k
IM <- 0                   ; reset IM for next sample
```

Figure 7: Pseudocode for the sample cache-decay controller.

three cache-decay schemes that give the best behavior overall: the fixed value of $T_d = 32$K for open-loop decay; and the values of 0.001 induced misses/cycle and PF = 0.3 for IMC and AMC respectively. Note that for open-loop decay, no information about induced misses is required and the tags can be decayed in addition to the data array. The IMC and AMC schemes, on the other hand, require information about which misses are induced, and the tags must be left active at all times.

We evaluate the three techniques in terms of normalized leakage savings, percent degradation in IPC, and turn-off ratio (TOR). Normalized leakage savings is simply the fraction of total cache leakage (in the tag and data arrays) that is saved after all overheads (induced fetches to L2, increased execution time, etc.) have been subtracted. Percent degradation in IPC is of course with reference to execution that does not use decay. And turn-off ratio is the average fraction of lines that are deactivated over the running time of the program. It quantifies, in a technology independent way, the leakage energy saved due to closing the cache lines. However, it does not indicate the overhead incurred in closing them and more importantly the effect of performance loss. For these reasons, our results show that TOR is not a useful metric for evaluating leakage-savings schemes.

	IMC	AMC	const best
crafty	74.4	28.2	64
gcc	60.7	34.3	32
go	130.4	138	32
ijpeg	47.3	73.9	32
li	174.2	131	32
parser	51	24.3	32
perl	150	130.8	128
vpr	37.3	5	16

Table 3: Mean Decay Intervals. "Const best" refers to the per-benchmark best value for open-loop decay.

	Open-loop	IMC	AMC
Mean IPC degradation	0.70%	0.30%	0.49%
Mean E_p (normalized)	30.39%	31.19%	32.32%

Table 4: Overall results for open-loop decay, IMC, and AMC.

5.1 Effectiveness for Data Cache

Figure 8 shows the IPC degradation, normalized leakage savings (E_p) and TOR for the open-loop, IMC and AMC controllers across the benchmarks. Table 4 summarizes the IPC-degradation and leakage-savings results. Despite the negligible differences among all three schemes' overall behavior that is apparent from Table 4, Figure 8 shows that from an average energy-profit standpoint, open-loop decay is actually slightly better than the feedback controllers for the majority of benchmarks. This is due to extra leakage savings from decaying the tags. The notable exception is *perl*, which prefers a much larger T_d (128 Kcycles) than the fixed value that was chosen (32 Kcycles, which happens to be the best constant interval for 6 of the 8 benchmarks). The feedback controllers are comparable in terms of the normalized energy savings. IMC usually performs better with respect to maintaining IPC, while AMC attains slightly better E_p.

Despite the better overall energy profits for open-loop decay, its non-adaptive nature also leads to very poor behavior for programs like *perl*, whose characteristics differ significantly from the "average" behaviour that dictated the fixed T_d. Adaptivity is essential to prevent such cases and ensure robust behavior. It does, however, come at a cost of as much as 3.5% in reduced leakage savings for some programs: so far, the only closed-loop schemes that have been devised—AMC and IMC—require that the tags stay powered on to provide the necessary feedback. The above results therefore demonstrate both the potential benefits (robustness and better adaptivity) and costs (conservative behavior and controller-implementation overheads) of closed-loop designs.

Better sensors that avoid this problem are an interesting area for future work.

To illustrate how the two controllers vary T_d during execution, Figure 9 plots, for two representative SPEC benchmarks, the decay intervals as selected by the IMC and AMC controllers over time. The behavior of *parser* is representative of programs for which IMC performs better in terms of energy saved compared to AMC. *Vpr* is representative of programs in which AMC performs better. In both the cases, IMC has a lower IPC degradation. Generally, for benchmarks like *parser* with very noisy behavior of misses per cycle we find that IMC is more stable than AMC, thereby giving more gains and demonstrating one advantage of using a linear controller like IMC rather than the non-linear step function that corresponds to AMC (see Figure 2).

IMC does not perform as well for benchmarks like *vpr*. This is partly because of the formulation of the setpoint: the IMC controller tries to bound the number of induced misses per cycle, whereas AMC tracks the true miss rate. In cases where the true miss rate is quite high, extra induced misses will have a minor impact on performance, and AMC is aggressive in selecting a shorter decay interval. The setpoint for IMC places possibly too much emphasis on the acceptable performance loss. So the decay interval selected is generally longer than that picked by AMC, at the expense of some leakage savings. To illustrate the different "philosophies" of these controllers, the mean T_d selected by IMC and AMC over the course of each simulation is summarized in Table 3. This table also shows the per-program best open-loop decay interval as a reference. Note that the best fixed decay interval is probably not actually a power of two; finding the best T_d is therefore a large search problem (another argument for feedback control).

Figure 10 shows the number of induced misses over the course of the program. The fixed line of 0.001 misses per cycle

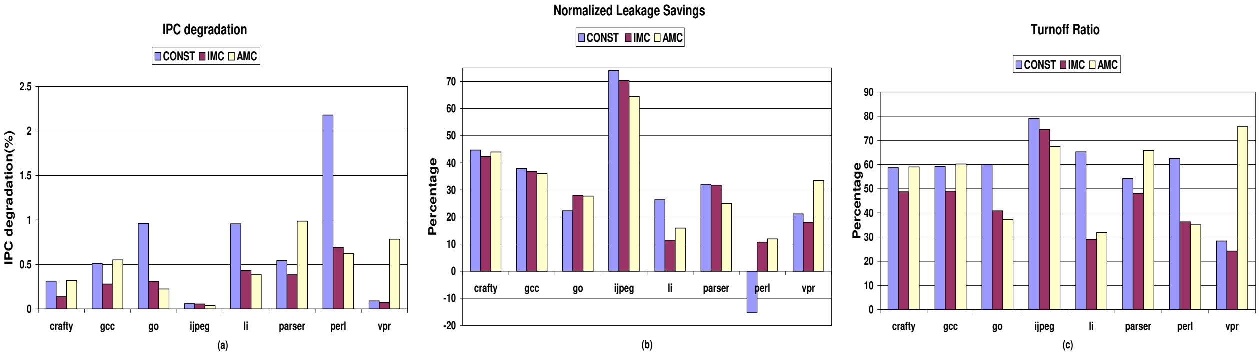

Figure 8: (a) IPC degradation, (b) Normalized leakage savings and (c) Turnoff ratio for the open-loop ("CONST"), IMC and AMC controllers.

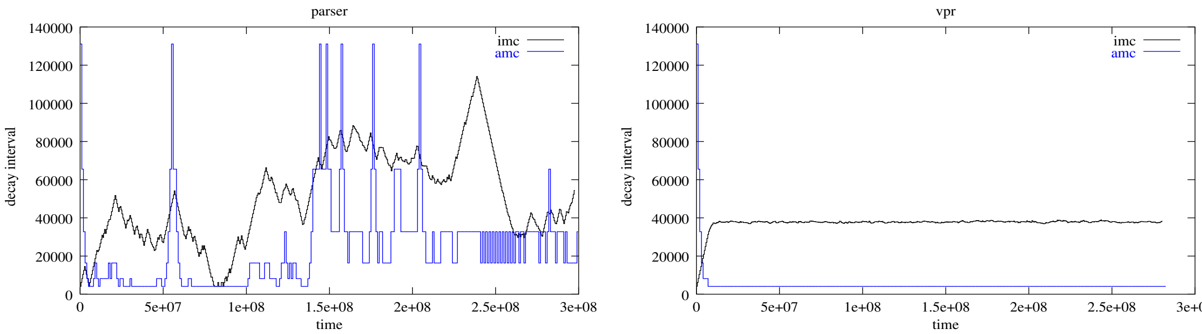

Figure 9: Decay interval over time for parser (left) and vpr (right). (IMC is the black, jagged line; AMC is the grey, squared-off line.)

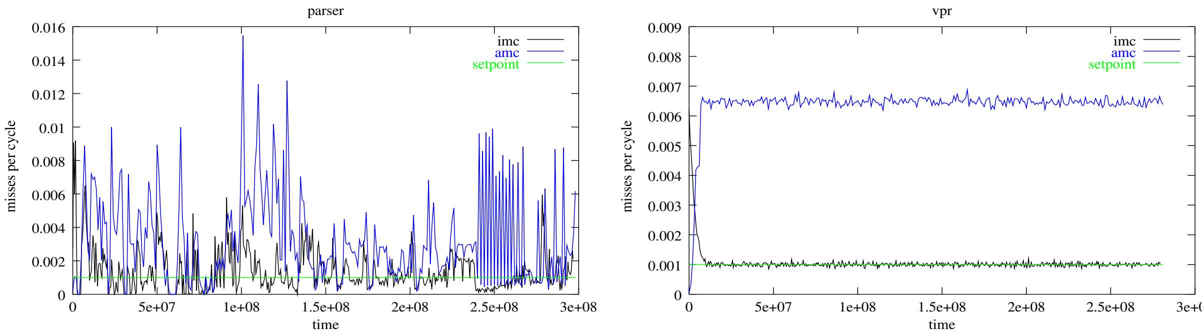

Figure 10: Induced misses per cycle for parser (left) and vpr (right). The IMC setpoint of 0.001 misses per cycle is the flat, gray line. IMC's observed miss rate is the line that stays very close to this setpoint. AMC appears to vary more widely because it is using a different setpoint and is merely included here to show their differing behavior.

denotes the IMC setpoint. We can see that the IMC controller tracks the setpoint very well. This is one of the reasons why the performance loss with IMC controller is almost always less than the performance loss with AMC controller or when using fixed decay intervals.

The choice of setpoint is a crucial issue in the design on a controller. With the IMC controller we have chosen a setpoint that targets a specified induced miss rate. From the Figure 10 we can see that the IMC controller tracks this setpoint very closely. Such a setpoint places more emphasis on the performance aspect (by controlling the additional induced misses) rather than the savings achievable. A different choice of the setpoint could be used when trying to improve energy savings at the expense of performance.

We also found that our setpoint lets us prove a nice bound on system behavior. Empirically, we observe that the IMC controller does well at what it is asked, namely holding the induced-miss rate to 0.001 per cycle. *Assuming* that this rate will successfully be enforced for other applications, IMC will never induce a performance loss of more than $0.001 * L * 100\%$ if the miss latency of the cache is L cycles. For the 11-cycle L1 miss latency we used, IMC never induces an IPC degradation of more than 1.1%—an observation borne out by our measurements.

A final comment is that our results also show that turn-off ratio is a poor metric for evaluating cache-decay schemes. Although an informative measure, it cannot be used in isolation, because it does not accurately represent the energy costs of decay. In some cases in Figure 8, the turnoff-ratio and the energy profit are inversely related.

6 Conclusions and Future Work

This paper has presented an overview of how to apply formal digital control to adaptive techniques that are based on sampling, and used cache decay as a running example. We derived an integral controller for cache decay that we call IMC, and compared its behavior to both open-loop decay and AMC, which is an adhoc feedback-control mechanism.

We found IMC easy to model, derive, and tune—easier than implementing AMC (even though its operation was already described in [27]) and tuning it for the fairest comparison. In terms of behavior, we found that open-loop decay is vulnerable to unanticipated or very diverse behaviors. Feedback control guards against these problems, but comes at some cost in tag energy (3.5%). This causes IMC and AMC to give slightly inferior energy savings for some benchmarks, although overall they are 3–6% better, and have the important benefit of reliable behavior. We argue that a slight reduction in energy savings for some applications is a reasonable cost to pay for security against bad open-loop behavior. As for comparing IMC against AMC, from the standpoint of energy savings and performance degradation, neither controller design is clearly superior for our limited set of benchmarks. We interpret this to mean that there is no reason *not* to use a formal controller in this case.

Adaptivity is a valuable tool for power and energy efficiency. Feedback control is valuable because it guides the adaptation to match program behavior to a desired behavior. As adaptivity becomes a major research area, the use of feedback control is certain to increase. We argue that feedback control should be designed using formal control theory. Even if not all the formalism is needed, portraying the system as a control loop and using standard control designs helps in understanding the system's behavior and ensures that the system is easy to analyze later should this be necessary.

We do recommend the stability analysis. In addition to choosing the gain for stability, the analysis tends to give a value for gain that gives good performance as well. The stability analysis therefore simplifies the task of tuning the controller.

We only implemented a simple integral controller. Even better results can probably be obtained using an optimization scheme that balances the marginal benefit of deactivating additional lines against the potential costs in terms of energy and performance loss.

Another open issue is the best way to choose a setpoint. Our current setpoint is performance-oriented. We were not able to fully explore this design space to find a way to balance energy and performance considerations. We also believe that a better choice of setpoint would have allowed us to prove a stronger bound on IPC degradation, without resorting to the assumption we used in Section 5.

Overall, we have shown that formal feedback-control techniques provide a range of benefits, and we hope that this paper will motivate its use in the architecture community for future work on adaptive designs.

Acknowledgements

This work is supported in part by the National Science Foundation under grant nos. CCR-0105626, CCR-0133634, and CCR-0098269. We would like to thank Stefanos Kaxiras, Margaret Martonosi, and Mircea Stan for their assistance and feedback, and Zhigang Hu for providing access to the simulator used in [10]. We would also like to thank the anonymous reviewers for their helpful comments.

References

[1] D. H. Albonesi. Dynamic IPC/clock rate optimization. In *Proc. ISCA-25*, pages 282–92, June 1998.

[2] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. HPCA-7*, pages 171–82, Jan. 2001.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. ISCA-27*, pages 83–94, June 2000.

[4] N. Dragone, A. Aggarwal, and L. R. Carley. An adaptive on-chip voltage regulation technique for low-power applications. In *Proc. ISLPED 2000*, pages 20–24, July 2000.

[5] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley, third edition, 1994.

[6] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, third edition, 1998.

[7] H. Hanson et al. Static energy reduction techniques for microprocessor caches. In *Proc. ICCD 2001*, pages 276–83, Sept. 2001.

[8] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. A control theoretic analysis of RED. In *Proc. IEEE INFOCOM*, Apr. 2001.

[9] S. Kaxiras. Personal communication, Oct. 2001.

[10] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proc. ISCA-28*, July 2001.

[11] 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.

[12] J. R. Leigh. *Applied Digital Control*. Prentice Hall, 1985.

[13] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Proc. IEEE Real-Time Technology and Applications Symp.*, June 2001.

[14] C. Lu, J. A. Stankovic, G. Tao, , and S. H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems J.*, Mar.-Apr. 2002. To appear.

[15] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *Proc. Int'l Conf. on Distributed Computing Systems*, Apr. 2001.

[16] S. McFarling. Combining branch predictors. Tech. Note TN-36, DEC WRL, June 1993.

[17] K. Nii et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proc. ISLPED 1998*, pages 293–98, Aug. 1998.

[18] SIA. *International Technology Roadmap for Semiconductors*, 1999.

[19] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proc. HPCA-8*, pages 17–28, Feb. 2002.

[20] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Trans. Computers*, 48(11):1260–81, Nov. 1999.

[21] M. R. Stan. Personal communication, Mar. 2002.

[22] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. http://www.specbench.org/osg/cpu2000.

[23] Standard Performance Evaluation Corporation. SPEC CPU95 Benchmarks. http://www.specbench.org/osg/cpu95.

[24] D. C. Steere et al. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. SOSP*, Feb. 1999.

[25] L. S. Y. Wong, S. Hossain, and A. Walker. Leakage current cancellation technique for low power switched-capacitor circuits. In *Proc. ISLPED 2001*, pages 310–15, Aug. 2001.

[26] 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 *Proc. HPCA-7*, Feb. 2001.

[27] Huiyang Zhou, Mark Toburen, Eric Rotenberg, and Thomas Conte. Adaptive mode control: A static-power-efficient cache design. In *Proc. PACT 2001*, Sept. 2001.