Adaptive Mode Control: A Static-Power-Efficient Cache Design

Huiyang Zhou, Mark C. Toburen, Eric Rotenberg, Thomas M. Conte
Department of Electrical and Computer Engineering
North Carolina State University
{hzhou, mctobure, ericro, conte}@eos.ncsu.edu

Abstract

Lower threshold voltages in deep sub-micron technologies cause more leakage current, increasing static power dissipation. This trend, combined with the trend of larger/more cache memories dominating die area, has prompted circuit designers to develop SRAM cells with low-leakage operating modes (e.g., sleep mode). Sleep mode reduces static power dissipation but data stored in a sleeping cell is unreliable or lost. So, at the architecture level, there is interest in exploiting sleep mode to reduce static power dissipation while maintaining high performance.

Current approaches dynamically control the operating mode of large groups of cache lines or even individual cache lines. However, the performance monitoring mechanism that controls the percentage of sleep-mode lines, and identifies particular lines for sleep mode, is somewhat arbitrary. There is no way to know what the performance could be with all cache lines active, so arbitrary miss rate targets are set (perhaps on a per-benchmark basis using profile information) and the control mechanism tracks these targets. We propose applying sleep mode only to the data store and not the tag store. By keeping the entire tag store active, the hardware knows what the hypothetical miss rate would be if all data lines were active and the actual miss rate can be made to precisely track it. Simulations show an average of 73% of I-cache lines and 54% of D-cache lines are put in sleep mode with an average IPC impact of only 1.7%, for 64KB caches.

1. Introduction

Power dissipation is becoming an important design constraint for high-performance processors. Projected increases in *static power dissipation* – power dissipated continuously, even when transistors are not switching – are particularly alarming. Borkar [2] estimates that with each new processor generation, leakage current and leakage power increase by a factor of 7.5 and 5.0, respectively. This is due to scaling down the threshold voltage of deep sub-micron technologies.

Caches consume a significant fraction of total die area, especially in high-performance embedded processors, e.g., 60% of the StrongARM die area is cache [1]. Therefore, among individual hardware components, caches potentially provide the greatest opportunity for static power reduction. Recently, two approaches have been proposed to reduce static power dissipation in caches: *DRI cache* [5] and *cache line decay* [6]. Both approaches exploit a circuit technique called *Gated-Vdd* [4], in which SRAM cells are isolated from the power and/or ground rails so that almost no static power is drawn. We refer to isolated cells as being in *sleep mode* or deactivated. A cache line in sleep mode loses its data and will cause a cache miss when re-accessed. However, caches tradeoff efficiency for robustness – caches are large enough to perform well on both large and small working sets. So, with careful performance monitoring, many cache lines can be deactivated most of the time with minimal performance impact.

DRI [5] dynamically activates/deactivates large groups of cache lines. The total number of sleep-mode cache lines is controlled by periodically examining the cache miss rate. The observed cache miss rate is compared to a pre-determined value, called the *miss bound*. If the observed miss rate is lower than the miss bound, then another large chunk of the cache is placed in sleep mode, since the observed miss rate is still within tolerated levels. If the observed miss rate exceeds the miss bound, then a large chunk of the cache currently in sleep mode is re-activated to help reduce the observed miss rate.

Cache line decay [6] activates/deactivates individual cache lines. The finer granularity with respect to DRI provides greater flexibility and is potentially more effective. A cache line is placed in sleep mode if it has not been accessed for a pre-determined amount of time, and is re-activated only when it is re-accessed.

A limitation of both DRI and cache line decay is their control mechanisms depend on arbitrary parameters that must be tuned per application to minimize the performance impact of static power reduction. In the case of DRI, miss bound is chosen based on the typical miss rate of an application, since ideally DRI should deactivate as many cache lines possible without exceeding the application's typical miss rate. So, DRI may require cache profiling. Cache line decay uses a different parameter, decay time (time that must elapse since the last access to a line before deactivating the line), but it too should be tuned per application. Evidence that decay time should be tuned is shown in Figure 1. By trial and error, a decay time was found for each benchmark that reduced performance by no more than 4% (the experiment was performed for a 64KB 4-way set-associative data cache). As can be seen in Figure 1, the tuned decay time varies from 14,000 cycles for *jpeg* to 98,000 cycles for *li*. In addition, tuning parameters does not always guarantee best results because (1) tuning reflects the behavior of profiled runs whereas any given run may behave differently and (2) static parameters cannot capture variations within a single run of the program.

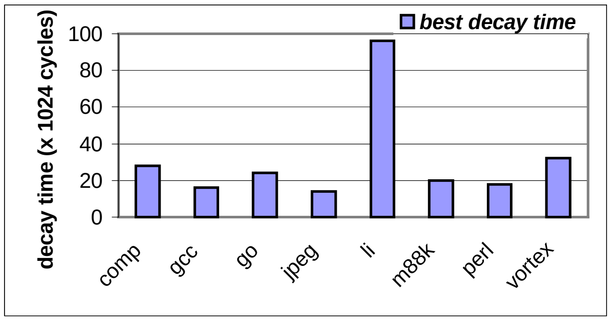

Figure 1. Per-benchmark cache line decay times, tuned to reduce performance by no more than 4%.

We propose deactivating only the data portion of cache lines, and not the tag portion. By keeping the entire tag store active, hardware can measure the hypothetical cache miss rate if we were to keep all lines active. Pre-determined parameters such as miss bound and decay time are no longer needed. Instead, hardware dynamically monitors the hypothetical miss rate using the tag store, and controls the total percentage of sleep-mode lines to achieve an actual miss rate that closely tracks the hypothetical miss rate. The method is able to self-adjust to variations among different applications and changes in cache requirements as a program executes.

Our method is called *Adaptive Mode Control* (AMC). Similar to cache line decay, AMC controls the mode (sleep vs. active) of individual cache lines according to a *turn-off interval*. The turn-off interval is the time that must elapse since the last access to a line before deactivating the line. The key difference is the turn-off interval is dynamically adjusted to ensure performance closely tracks the performance of an equivalent cache without sleep mode. That is, the turn-off interval is variable, and its value is periodically adjusted based on the number of extra misses caused by sleep-mode cache lines. Because the tags remain active, hardware is able to distinguish between two types of misses.

1. *Ideal miss*: This is a miss in the tag store, i.e., the line is not in the cache in either active or sleep mode. This miss would have occurred in a conventional cache of equivalent complexity.
2. *Sleep miss*: A sleep miss occurs when there is a hit in the tag store but the data portion of the cache line is in sleep mode. The line is in the cache, but it is unusable and results in a cache miss.

A variety of simple control systems are possible. In this paper, we develop an effective control system that examines the ratio of sleep misses to ideal misses. If the ratio is “too small”, AMC can be more aggressive in deactivating cache lines, so the turn-off interval is reduced. If the ratio is “too large”, AMC must be more conservative in deactivating cache lines, so the turn-off interval is increased. If the ratio is “just right”, the turn-off interval is kept the same.

Our results show that for the SPECint95 benchmarks, an average of 73% of instruction cache lines and 54% of data cache lines can be deactivated during program execution time with an average performance loss of only 1.7%, for 64KB instruction and data caches.

The remainder of this paper is organized as follows. Section 2 describes the AMC architecture and hardware mechanism for controlling the turn-off interval. The SRAM cell circuit with sleep mode, developed by others and used in this work, is reviewed in Section 3. The simulation methodology and results are presented in Sections 4 and 5, respectively. Section 6 discusses related work, and Section 7 concludes the paper.

2. Adaptive Mode Control

2.1. AMC Cache Architecture

In order to make efficient use of sleep-mode-capable SRAMs, we need the ability to monitor accesses to individual cache lines. We do this by associating a counter with each tag in the tag store, as shown in Figure 2 for a direct mapped cache (the same applies for set-associative caches). These counters are called *Line Idle Counters* (LICs), because they keep track of how long a cache line has not been accessed. If a cache line has been idle (i.e., not accessed) for a sufficient period of time, it will be placed in sleep mode. Below, we first describe how LICs are maintained (reset and incremented), and then describe how LICs are monitored to deactivate lines (LIC compared to a threshold).

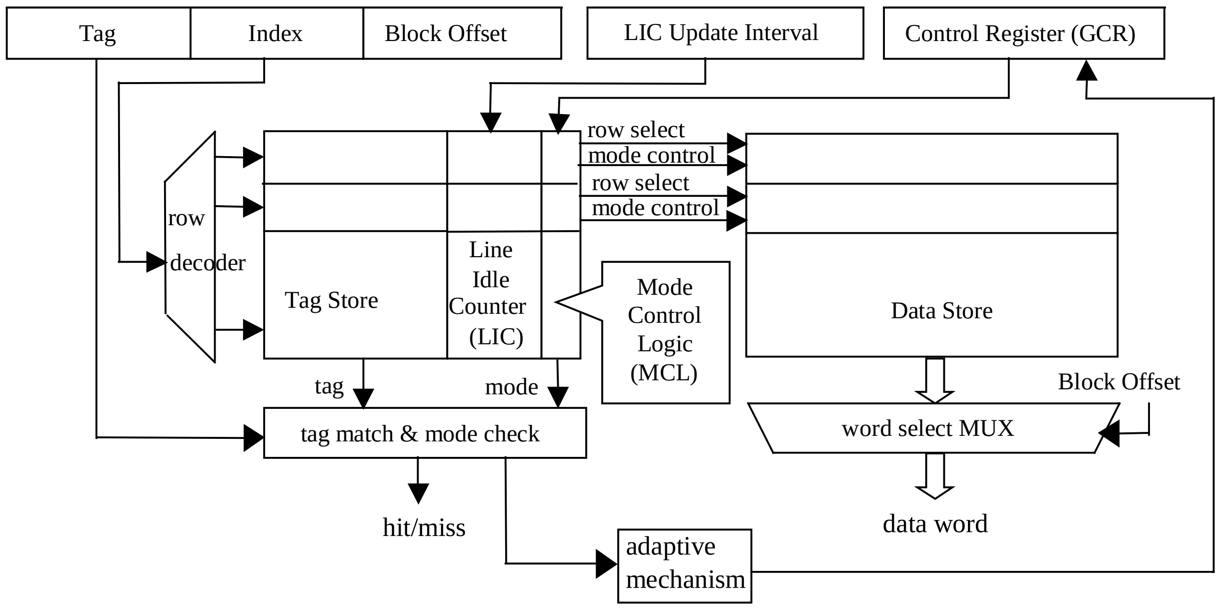

Figure 2. AMC cache architecture (direct mapped cache shown).

A LIC is reset when the corresponding line is accessed. All LICs in the tag store are simultaneously incremented after a certain number of cycles have elapsed, called the *LIC update interval*. The LIC update interval is constant and implementation-dependent, the choice of which depends on a straightforward tradeoff between the hardware area/power overhead of LIC counters and aggressiveness in deactivating cache lines. A sufficiently long LIC update interval (1) results in a small number of bits for each LIC counter since a single increment represents a longer time interval and (2) reduces the frequency of incrementing the counters, keeping their dynamic power contribution quite low. However, if the LIC update interval is too long, then AMC is slower to deactivate cache lines, squandering opportunities to save static power [6]. Detailed analysis in Section 5.4 shows a LIC update interval of 2048 cycles yields effective results with small counter area overhead and negligible dynamic power overhead (LICs increment infrequently – once every 2048 cycles) [23].

Also associated with each tag/LIC is a small comparator logic block called the *Mode Control Logic* (MCL), as shown in Figure 2. The MCL associated with each LIC compares the LIC value to the *turn-off interval* stored in the *Global Control Register* (GCR). If the LIC value is greater than or equal to the turn-off interval in the GCR, the MCL will place the corresponding data line into sleep mode (this is achieved with a control wire, labeled “mode control” in Figure 2). Otherwise, the line remains in active mode.

Note, for write-back data caches, dirty data needs to be written back to the L2 cache/main memory before a line is placed in sleep mode because data is not retained in sleep mode. This may impact performance either positively or negatively; in some sense, the LIC/MCL logic is an implementation of eager writeback [22] which suggests it is possible for performance to improve.

Finally, miss detection is modified slightly in an AMC cache because there are two types of misses (as described in Section 1), *ideal misses* and *sleep misses*. An ideal miss occurs when the tag(s) do not match. A sleep miss occurs when a matching tag is found but the data portion is in sleep mode. A sleep miss is handled like any other cache miss, i.e., data must be fetched from the L2 cache and the cache line is re-activated to hold the fetched data. So, ideal misses are misses that would occur in a conventional cache of equivalent size/associativity and sleep misses are additional misses introduced by AMC. The goal of AMC is to maximize the number of deactivated lines while minimizing the number of sleep misses.

The turn-off interval, stored in the GCR, determines how aggressively cache lines are deactivated. When the GCR is too small, many soon-to-be-accessed cache lines will prematurely deactivate, resulting in many sleep misses. In this case, performance suffers and dynamic power (due to miss servicing) may increase. On the other hand, if the GCR is too large, AMC is slow to deactivate lines. Lines that are unused before being evicted are not deactivated early enough to reap any static power savings. It is the job of the *adaptive mechanism* shown at the bottom of Figure 2 to monitor the overall system and tune the GCR to achieve maximum static power savings with little or no performance loss.

2.2. Adaptive Mechanism

The fact that cache tags are never put into sleep mode allows hardware to separate overall misses into those that would have occurred regardless of a line’s sleep/active

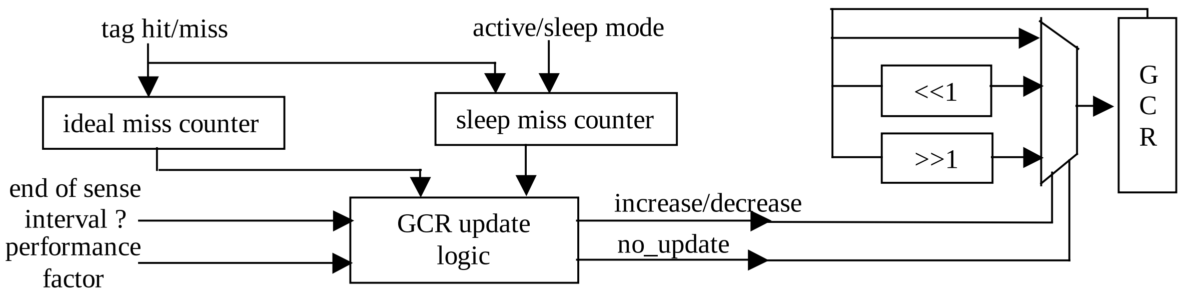

Figure 3. Adaptive mechanism for dynamically updating the turn-off interval stored in the GCR.

status (ideal misses), and extra misses specifically caused by sleep-mode lines (sleep misses). (Note: overall misses = ideal misses + sleep misses.)

Ideal and sleep misses are counted during a *sense interval*, a fixed period of time. At the end of the sense interval, the adaptive algorithm examines the gathered miss counts and updates the GCR. Then, the miss counters are reset and counting begins anew for the next sense interval.

The adaptive mechanism is shown in Figure 3. The ideal miss counter is incremented when there is a tag miss (tag_miss). The sleep miss counter is incremented when there is a tag hit *and* the data is in sleep mode (tag_hit & data_sleep). The inputs to the GCR update logic are (1) the ideal miss count, (2) the sleep miss count, (3) the *end-of-sense-interval* signal, and (4) the *performance factor* (PF). The end-of-sense-interval signal simply indicates when the GCR update logic should examine the miss counts and update the GCR. We will show in Section 5.4 that the duration of the sense interval has little impact on AMC results (both performance and power savings).

The diagram in Figure 4 illustrates how our proposed adaptive mechanism works. The bold line closest to the x-axis represents the number of ideal misses. Thus, the distance between the x-axis and the first bold line is labeled “ideal misses”. Sleep misses are considered *error* in the system since they add to the ideal misses. Targeting an error of 0 with our control system is rather conservative (although not impossible), so we show a second bold line above the first and the distance between the two bold lines is the amount of error – this distance is labeled “target error”. Moreover, rather than define the target error as an absolute number, it makes more intuitive sense to define the target error as a fraction/percentage of

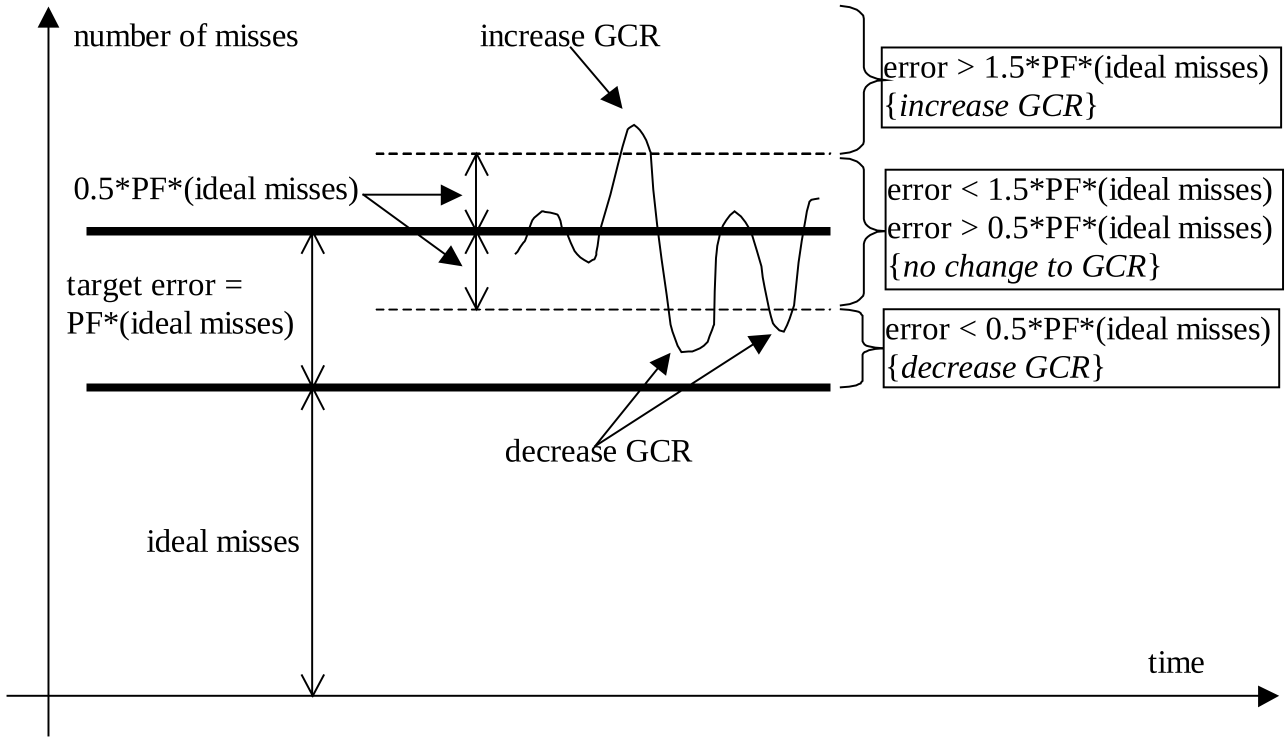
number of misses
increase GCR
error > 1.5*PF*(ideal misses)
{increase GCR}
0.5*PF*(ideal misses)
error < 1.5*PF*(ideal misses)
error > 0.5*PF*(ideal misses)
{no change to GCR}
target error = PF*(ideal misses)
error < 0.5*PF*(ideal misses)
{decrease GCR}
decrease GCR
ideal misses
time

Figure 4. Diagram explaining control system in terms of target error. In AMC, error = sleep misses.

the base quantity, in this case the number of ideal misses (like tolerances in discrete resistors). We call this fraction the performance factor, PF. So, the target error (or targeted number of sleep misses) is equal to PF*(ideal misses).

Now that there is a target error, we can make a control system. If the observed error is less than the target error, the GCR is decreased (more lines go to sleep mode, creating more error but getting more power savings). If the observed error is greater than the target error, the GCR is increased (fewer lines go to sleep mode, bringing the error back down). This approach is too simplistic because it is unlikely the target error will ever be met exactly, so the system reacts to even the smallest deviations.

To improve on the above, we define a target error *range* that is centered around the target error. This range is shown in Figure 4 with two dashed lines above and below the top bold line. So, we still target an error of PF*(ideal misses), but we only need to get within a certain range of it to not change the GCR. The range is the same size as the target error itself, or PF*(ideal misses), so the two dashed lines are 0.5*PF*(ideal misses) above and below the top bold line, as labeled in Figure 4. Therefore, the target error range is expressed as: PF*(ideal misses) ± 0.5*PF*(ideal misses). Making the target error and its tolerance similar in magnitude is intuitively appealing and, as will be described below, it also simplifies the hardware implementation of the algorithm – it uses only a few shifts, adds, and compares as a result.

The GCR update algorithm (shown in Figure 4 and codified in Figure 5) is based on the target error range described above. The GCR is decreased when the number of sleep misses is less than 0.5*PF*(ideal misses). The GCR is decreased via right-shifting it by one bit (see Figure 3). This reduces the turn-off interval, in turn more aggressively deactivating cache lines, since the error is below the target error range. As an example, the GCR is decreased twice for the miss curve shown in Figure 4, since the curve dips below the lower dashed line twice. The GCR is increased when the number of sleep misses exceeds 1.5*PF*(ideal misses). The GCR is increased via left-shifting it by one bit (see Figure 3). This increases the turn-off interval, in turn less aggressively deactivating cache lines, since the error is above the target error range. For the example miss curve in Figure 4, the GCR is increased once where the curve peaks above the upper dashed line. If the number of sleep misses is within the target error range, the GCR is not changed.

```
if ((sleep misses) < ((ideal misses)*0.5*PF)) {
   decrease GCR: shift GCR right by one bit
}
else if ((sleep misses) > ((ideal misses)*1.5*PF)) {
   increase GCR: shift GCR left by one bit
}
else {
   do not change GCR
}
```

Figure 5. The GCR update algorithm.

The GCR update algorithm can be implemented in hardware using shifts, adds, and compares (subtracts). PF is set to a power-of-2 such as 2, 1, ½, ¼, etc. Therefore, 0.5*PF is a power-of-2 fraction, and the right-hand side of the first if-expression in Figure 5 is implemented via a right-shift of the ideal miss counter. The right-hand side of the second if-expression is implemented in two steps: 1.0*PF*(ideal misses) is implemented as a right-shift of the ideal miss counter, and this result is added to the result computed in the first if-expression to obtain 1.5*PF*(ideal misses). The two if-conditions are then evaluated via two magnitude-comparators, where the above quantities and the number of sleep misses are operands.

Due to the nature of our negative feedback algorithm, the average ratio of sleep misses to ideal misses settles close to the desired PF [23].

It should be noted that any control system necessarily has pre-defined constants, PF in the case of AMC, miss bound and size bound in the case of DRI [5], and decay time in the case of cache line decay [6]. All of these schemes dynamically track performance. AMC's distinction is it tracks a dynamic and accurate performance target (ideal misses) instead of a pre-defined and potentially less accurate performance target. PF, albeit a pre-defined parameter, is a multiplicative coefficient and determines how *closely* AMC performance tracks hypothetical performance.

3. Review of SRAM Cell with Sleep Mode Capability

Recently, researchers in the VLSI community have proposed several techniques for reducing the static power dissipated in memory cells due to leakage current [3,4,8-10]. The design assumed in this work is shown in Figure 6.

The cell can be isolated from the power and ground rails. Two additional nodes, *virtual vdd* (vvdd) and *virtual gnd* (vgnd), are introduced, and the voltage at these two nodes is controlled by transistors Q1 and Q2, which are high-V_t (threshold voltage) or long-channel devices. When the circuit is in active mode, both Q1 and Q2 are on and the circuit operates as usual. When in the sleep mode, Q1 and Q2 are turned off and the leakage current through the SRAM cell is reduced dramatically due to the transistor stacking effect [7]. Several variants of this implementation and the implications on power, area, and performance are discussed in [4].

The cell can be improved by placing diodes between vgnd-gnd and vvdd-vdd, retaining data in sleep-mode [8]. Sleep-mode lines hit, but with long latency to turn on. So, AMC and this cell mutually benefit: AMC minimizes slow sleep-hits and the new cell eliminates the need to consume power fetching from the L2 cache.

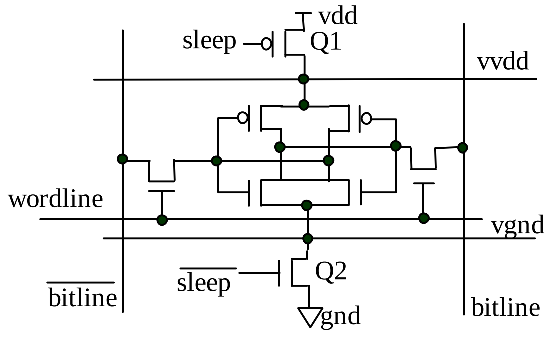

Figure 6. SRAM cell with sleep mode support.

4. Methodology

This section describes our simulation environment, including the underlying processor architecture, benchmarks, and baseline AMC parameters.

4.1. Simulation Environment

We developed a cache simulator that fully models the AMC architecture and integrated it into a timing simulator developed using the Simplescalar toolset [16]. The underlying processor organization is based on the MIPS R10000 processor, configured as indicated in Table 1. The SPECint95 benchmarks, listed in Table 2, were run to completion.

Table 1. Processor configuration.

Instruction Cache	Size = 16/32/64 kB
	Associativity = direct-mapped/4-way
	Replacement = LRU
	Line size = 16 instructions (64 bytes)
	Miss penalty = 12 cycles
Data Cache	Size = 16/32/64 kB
	Associativity = direct-mapped/4-way
	Replacement = LRU
	Line size = 64 bytes
	Miss penalty = 14 cycles
Superscalar Core	Reorder buffer: 64 entries
	Dispatch/issue/retire bandwidth: 4-way superscalar
	4 fully-symmetric function units
	Data cache ports: 4
Execution Latencies	Address generation: 1 cycle
	Memory access: 2 cycles (hit in data cache)
	Integer ALU ops = 1 cycle
	Complex ops = MIPS R10000 latencies

Table 2. Benchmarks.

Benchmark	Input dataset	Instruction Count
compress	compress95.ref	24 million
gcc	-O3 genrecog.i –o genrecog.s	117 million
go	9 9	133 million
jpeg	Vigo.ppm	166 million
li	Test.lsp (queens 7)	202 million
m88ksim	-c < ctl.in (dcrand.big)	120 million
perl	scrabble.pl < scrabble.in (dictionary)	108 million
vortex	vortex.in (persons.250, bendian.*)	101 million

4.2. Default AMC Parameters

Throughout the remainder of the paper we use default values for three primary AMC parameters: the performance factor (PF) is set at ½, the sense interval is set at 1 million cycles, and the LIC update interval is set at 2048 cycles. The combination of these default settings provides the best trade-off between static power savings and performance in our studies. In Section 5.4 we examine the impact of varying each of these parameters individually on static power and performance.

5. Results

In this section, we apply AMC to L1 instruction caches (I-caches) and data caches (D-caches), separately and together. Specifically, we examine the performance impact of AMC and the percentage of cache lines that are placed in sleep mode. Performance is measured as instructions-per-cycle (IPC), and we present the *% IPC degradation*. The percentage of cache lines that are placed in sleep mode is called the *turn-off ratio*. Turn-off ratio is measured by recording the fraction of cache lines in sleep-mode each cycle, and averaging over all cycles. We assume static power savings is proportional to the turn-off ratio.

5.1. AMC Instruction Caches

We studied 16kB, 32kB, and 64kB instruction caches, for each of direct-mapped and 4-way set-associativity. The D-cache in all experiments is 64kB 4-way set-associative without AMC.

Figure 7 shows % IPC degradation over all benchmarks and I-cache configurations. The primary result, as expected, is that performance is never degraded by more than 6.5%, and performance is never worsened by more than 3% on average.

Figure 7 also shows that the performance impact of AMC is sensitive to I-cache size and associativity. Firstly, performance impact is less with higher associativity. There are fewer ideal misses with 4-way set-associative caches than with direct-mapped caches, and our control system targets a number of sleep misses proportional to the number of ideal misses via PF. This is a small price we pay for using a multiplicative coefficient rather than an arbitrarily set, absolute bound on the number of sleep misses. In Section 5.4, we study the effects of varying PF on both static power and performance.

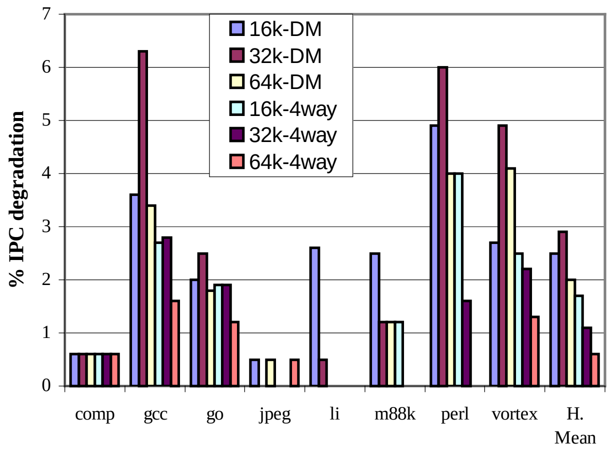

Figure 7. AMC I-cache: performance degradation.

Secondly, with 4-way set-associative caches, increasing cache size results in less performance degradation (except for a small deviation in *gcc* for the 32kB cache), for the same reason just described: larger caches have fewer ideal misses and our control system will generate fewer sleep misses as a result.

Thirdly, for direct mapped caches, the performance trend with cache size is somewhat more unusual than with 4-way set-associative caches. For *gcc*, *go*, *perl*, and *vortex*, performance degradation with a 16kB direct-mapped cache is lower than with a 32kB direct-mapped cache, yet we would expect it to be higher: more ideal misses with 16kB than with 32kB, therefore, more sleep misses too. Ideal miss rate is higher, e.g., for *gcc*, 10% for 16kB and 6% for 32kB. But the sleep miss rate does not behave as expected. For *gcc*, the 16kB cache has a 2.2% sleep miss rate and the 32kB cache has a 2.7% miss rate. We conclude that, with too small a cache to fit the working set (16kB), lines are almost always re-accessed before having a chance to turn-off. For these results, the minimum turn-off interval is 4K cycles. Most lines in the 16kB cache are re-accessed within 4K cycles. If the turn-off interval could dip below 4K cycles, 16kB IPC degradation and turn-off ratio would both be higher. But, it is encouraging that the lower bound on turn-off interval also bounds the sleep miss rate when ideal miss rate becomes too high (a built-in safety mechanism). Interestingly, in *gcc* (our largest benchmark), the 16kB 4-way set-associative cache shows slightly the same trend as the 16kB direct-mapped cache.

Figure 8 shows the I-cache turn-off ratio for all benchmarks and cache configurations. The first conclusion is that AMC provides significant static power savings, from 40% (16kB 4-way cache) to 77% (64kB direct-mapped cache) turned-off cache-lines, justifying the relatively small performance degradation.

Two other trends are evident from Figure 8. Turn-off ratios (1) decrease with decreasing cache size and (2) decrease with increasing associativity. The first trend is expected since, as cache size decreases, an application's working set consumes a larger percentage of the cache. The second trend is less intuitive and can be explained via a contrived example. Consider a direct-mapped cache and a fully-associative cache, and suppose all accesses map to the same line in the direct-mapped cache but obviously not so in the fully-associative cache. All but that single line will be turned off in the direct mapped cache; none of the lines in the fully-associative cache will be turned off. So, even though a 32kB 4-way cache is effectively larger than a 32kB direct mapped cache, that does not mean the turn-off ratio will be higher for the associative cache. In fact, the *utilization* of the 32kB space improves with associativity so fewer lines are deactivated. It is precisely because the direct mapped cache performs substantially worse to begin with that deactivating more of it is possible.

Finally, from Figure 8 it is apparent that AMC is able to dynamically adjust to different behavior among benchmarks. Specifically, the turn-off ratio varies substantially while performance degradation is kept fairly low across all benchmarks. Turn-off ratios for a 32kB 4-way set-associative instruction cache are: 94% (*compress*), 41% (*gcc*), 55% (*go*), 73% (*jpeg*), 66% (*li*), 56% (*m88ksim*), 48% (*perl*), and 37% (*vortex*).

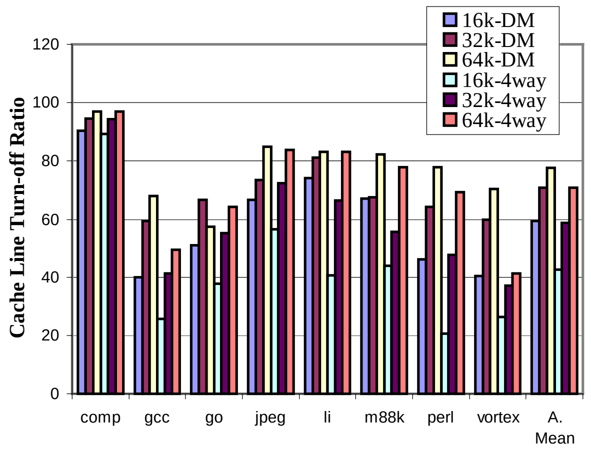

Figure 8. AMC I-Cache: Turn-off Ratio.

Another interesting result can be deduced by examining Figures 7 and 8 together. Notice *gcc*, *go*, *perl*, and *vortex* have relatively higher performance degradation and relatively lower turn-off ratio, compared to the other four benchmarks. Again, these benchmarks have relatively larger working sets.

5.2. AMC Data Caches

We also studied 16kB, 32kB, and 64kB direct-mapped and 4-way set-associative data caches. The I-cache in these experiments is 64kB 2-way set-associative without AMC.

Figure 9 shows the % IPC degradation for each benchmark and cache configuration. The AMC D-cache degrades IPC as much as 8.3% among individual benchmarks and 4.6% on average, compared with 6.5% and 3%, respectively, for AMC I-caches. D-cache ideal miss rates are higher, therefore, its sleep miss rates are higher.

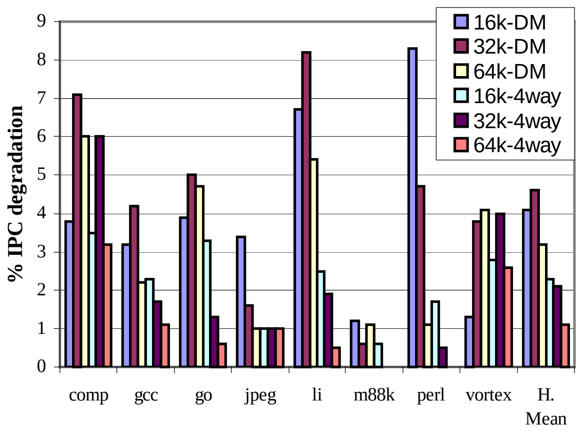

Figure 9. AMC D-cache: performance degradation.

From Figure 9, the AMC D-cache has the same performance trends with cache size and set-associativity as the AMC I-cache in the previous section. First, IPC degradation decreases with increasing set-associativity. Second, IPC degradation decreases with cache size, although again we see the same phenomenon with the 16kB caches. That is, the 16kB direct mapped cache is too small for *compress*, *gcc*, *go*, *li*, and *vortex*, such that the turn-off interval saturates at its lower limit and performance degrades less than expected. For *compress* and *vortex*, this is even true for the 16kB 4-way set-associative cache. (We even see the same trend for a 32kB direct mapped cache in *vortex*.)

Figure 10 shows the D-cache turn-off ratio for all benchmarks and cache configurations. The main conclusion is that AMC D-caches provide somewhat less static power savings than AMC I-caches, from 38% (16kB 4-way cache) to 75% (64kB direct-mapped cache).

Again, similar to AMC I-caches, we see from the average results in Figure 10 that AMC D-cache turn-off ratios tend to decrease with decreasing cache size and decrease with increasing associativity. However, for D-caches, there are more deviations from these average trends among individual benchmarks.

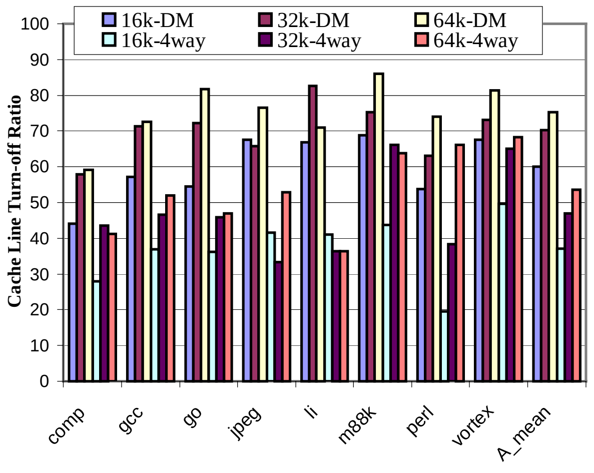

Figure 10. AMC D-cache: Turn-off Ratio.

5.3. AMC I-cache and D-cache

AMC can be applied simultaneously to both the instruction cache and data cache, with significant static power savings but only minor performance loss. For a 64kB 2-way I-cache and 64kB 4-way D-cache, we get a turn-off ratio of 73% for the I-cache and 56% for the D-cache. The performance degradation is only 1.8%, which interestingly turns out to be equal to the sum of the performance degradations measured individually in Sections 5.1 and 5.2. Note, PF = ½ for both D-cache and I-cache, and each has a separate GCR.

5.4. Sense interval, performance factor, and LIC update interval

The AMC sense interval determines how often the GCR should be updated. In our studies we used a fixed value of 1 million cycles for both I-caches and D-caches. Although a time-varying value capable of detecting distinct execution phases is ideal, our studies do not show a significant difference when the sense interval is varied from 250000 to 4000000 cycles. The variation in performance is less than 1% and the variation in power saving is less than 2% for I-caches and D-caches, with *compress* being the only exception. In *compress*, the performance variation is 1.3% (IPC from 1.49 to 1.51) and the variation in power savings in the D-cache is 9% (from 48.1% to 39.0%).

The performance factor PF determines the trade-off between performance degradation and static power savings by controlling the sleep-to-ideal miss ratio. A smaller PF implies that we are more sensitive to an increase in sleep misses. In the extreme, by setting the performance factor to zero, we effectively turn AMC off. Table 3 shows results for each benchmark using a 64kB 2-way I-cache and 64kB 4-way D-cache, in which we varied PF from 1/8 to 1 (again, PF is the same for both caches

Table 3. Simulation results with different performance factors (PFs).

Benchmarks	IPC				I-cache turn-off ratio			D-cache turn-off ratio		
	Ideal	PF=1/8	PF=1/4	PF=1	PF=1/8	PF=1/4	PF=1	PF=1/8	PF=1/4	PF=1
Compress	1.56	1.53	1.53	1.49	97.0%	97.0%	97.0%	35.0%	36.3%	46.8%
Gcc	1.84	1.83	1.81	1.74	45.2%	52.9%	66.0%	36.5%	44.3%	60.3%
Go	1.64	1.63	1.62	1.58	15.8%	31.3%	73.4%	40.1%	42.2%	53.3%
Jpeg	1.97	1.96	1.95	1.94	83.8%	83.8%	83.9%	48.0%	51.9%	62.3%
Li	2.19	2.18	2.18	2.18	82.9%	82.9%	83.0%	36.4%	36.4%	36.4%
M88ksim	1.74	1.74	1.74	1.74	78.1%	78.3%	78.7%	63.7%	63.8%	79.7%
Perl	1.91	1.91	1.91	1.91	70.2%	70.3%	70.3%	65.8%	66.1%	66.7%
Vortex	2.35	2.32	2.30	2.21	40.5%	43.6%	57.6%	45.7%	55.6%	76.8%
Average	1.868	1.855	1.848	1.817	64.2%	67.5%	76.2%	46.4%	49.6%	60.3%

and each cache has its own GCR). As expected, increasing PF tends to decrease IPC and increase turn-off ratio. IPC for *li* and *m88ksim*, however, is insensitive to PF. The ideal miss rates of both caches for *li* and *m88ksim* are quite low, so both GCRs saturate at the maximum value most of the time (to keep sleep miss rate also very low).

Finally, we study the impact of LIC update interval. As discussed in Section 2.1, a finer interval granularity provides more opportunity for cache lines to be put into sleep mode. However, a coarser interval granularity results in smaller area cost and dynamic power consumption of the LICs. Simulations showed variations of 0.5% and 2.0% in IPC and turn-off ratio, respectively, as the LIC granularity varies from 256 to 4096 cycles. We conclude a 2048-cycle interval provides a good trade-off between counter overhead (area and dynamic power) and aggressiveness in deactivating cache lines.

6. Related work

Recently, as power has become a first-order design constraint, there has been a deluge of research in architectural power modeling and optimization of on-chip caches. Several techniques have been proposed to reduce the switching power of on-chip caches. With support from the compiler, selective cache ways [11] enables an appropriate number of ways based on the cache requirements of the current application. The unused ways are disabled by the cache controller through the Cache Way Select Register (CWSR). The L-Cache [12] and Filter cache [13] attempt to reduce L1 cache activity by placing a small L0 cache between the L1 and the processor. With the compiler taking the responsibility of code modification and allocation of instructions into the L-Cache, much smaller performance degradations result as compared to the Filter cache. Block buffering [14] is similar in concept, but, instead of an additional cache level, it places recently requested words into a block buffer inside the cache. With the use of two-phase clocking, the additional access latency can be minimized. Sub-banking in the data array [14] and multiple-divided modules (MDM) [15] also reduce the power consumption by accessing only part of the cache line. In addition to these techniques, several analytical energy models [14, 18, 19] have been proposed to estimate and evaluate cache power and power saving techniques.

The primary goal of the approaches discussed previously is to reduce dynamic power dissipation. The DRI I-cache [5], as mentioned in Section 1, is a mechanism for reducing static power consumption by dynamically resizing and turning-off unused sections by way of the Gated-Vdd technique [4]. As the I-cache size changes over time, an index re-mapping mechanism is necessary which incurs a resizing penalty. In order to obtain optimal power-performance trade-off results, the control parameters, such as miss bound and size bound, must be pre-tuned for different applications. Cache Line Decay [6] targets static power reductions through the use of the Gated-Vdd technique by turning off individual cache lines that have not been accessed for some pre-defined interval – the decay interval. Since the decay interval is statically fixed, it cannot be updated dynamically to accommodate changes in cache requirements within and across applications.

Cache Line Decay was recently improved upon to make it more dynamic, without keeping the tag store active [24]. An interesting future study would be to compare the two alternative dynamic approaches, AMC cache and the Cache Decay approach.

In this paper, we only presented turn-off ratio results. A detailed static and dynamic power analysis of the AMC cache can be found in the companion technical report [23].

7. Conclusions

We proposed a microarchitecture technique that dynamically adapts to evolving cache requirements in order to conserve static power while maintaining performance. The main contributions of this study are as follows.

- The tag store is always kept active. This enables hypothetical performance without sleep mode to be determined and used to control real performance. Dynamically monitoring hypothetical performance is an improvement over setting arbitrary and static performance targets.

- We proposed a control system that keeps the number of sleep misses within a certain factor of ideal misses. Using a relative factor instead of an arbitrary, absolute number is a key contribution.
- We presented extensive results, including multiple I-cache and D-cache configurations and sensitivity to AMC parameters. Previously unknown, interesting results emerged. Just one interesting example is higher associativity results in lower cache turn-off ratios. This was initially counter-intuitive but the explanation is associative caches utilize a fixed amount of space better than direct-mapped caches.
- We demonstrated that AMC is overall a very effective means for improving static-power-efficiency in caches while maintaining good performance. Our overall results show that an average of 73% of I-cache lines and 54% of D-cache lines can be turned off with only a 1.8% performance loss.

8. Acknowledgments

This research was supported by generous funding and equipment donations from Compaq, Intel, and Ericsson, and by NSF CAREER grant No. CCR-0092832.

9. References

[1] J. Montanaro, *et al*, "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor." *Digital Technical Journal*, vol. 9, Digital Equipment Corporation, 1997.

[2] S. Borkar, "Design Challenges of Technology Scaling." *IEEE Micro*, 19(4), July 1999.

[3] M. Margala, "Low Power SRAM Circuit Design", *IEEE Int. Workshop on Memory Technology, Design, and Testing*, 1999.

[4] M. Powell, S-H. Yang, B. Falsafi, K. Roy and T. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories." *Proc. of the Int. Sym. on Low Power Electronics and Design (ISLPED)*, 2000.

[5] S-H. Yang, M. Powell, B. Falsafi, K. Roy and T. Vijaykumar, "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-caches", *Proc. of the Int. Sym. on High Performance Computer Architecture (HPCA)*, Jan. 2001.

[6] S. Kaxiras, Z. Hu, G. Narlikar, and R. Mclellan, "Cache-line decay: A Mechanism to Reduce Cache Leakage Power", *IEEE workshop on Power Aware Computer Systems*, 2000.

[7] Y.Ye, S. Borkar and V. De, "A New Technique For Standby Leakage Reduction in High Performance Circuits." *IEEE Sym. on VLSI Circuits*, 1998.

[8] K. Noii, et.al., "A Low Power SRAM using Auto-Backgate-Controlled MT—CMOS", *Proc. of the Int. Sym. on Low Power Electronics and Design (ISLPED)*, 1998.

[9] S. Shigematsu, et. al., "A 1-v High Speed MTCMOS Circuit Scheme for Power-Down Application Circuits", *IEEE Journal of Solid-State Circuits*, vol. 32, 1997.

[10] T. Kuroda, et. al., "A 0.9v, 150MHz 10mW, 4mm^2, 2-D Discrete Cosine Transform Core Processor with Variable Threshold-Voltage Scheme", *IEEE Journal of Solid-Srate Circuits*, vol. 31, 1996.

[11] D. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation", *Proc. of the 32nd Annual IEEE/ACM Int. Sym. on Microarchitecture (MICRO 32)*, Nov. 1999.

[12] N. Bellas, I. N. Hajj, C. D. Polychronopoulos, and G. Stamoulis, "Architectural and Compiler Techniques for Energy Reduction in High-Performance Microprocessors", *IEEE Trans. On VLSI Systems*, 8(3), 2000.

[13] J. Kin, M. Gupta, and W. Mangione-Smith, "The filter cache: An energy efficient memory structure", *Proc. of the 30th Annual IEEE/ACM Int. Sym. on Microarchitecture (MICRO 30)*, 1997.

[14] M. B. Kamble and K. Ghose, "Analytical Energy Dissipation Models for Low Power Caches", *Proc. of the Int. Sym. on Low Power Electronics and Design (ISLPED)*, Aug. 1997.

[15] U. Ko and P.T. Balsara, "Characterization and Design of A Low-Power, High-Performance Cache Architecture", *Int. Sym. on VLSI Technology, Systems, and Applications*, 1995.

[16] D. Burger and T. M. Austin, "The Simplescalar Tool Set Version 2.0", Technical Report, Computer Science Department, University of Wisconsin-Madison, 1997.

[17] N. Bellas, I. Hajj, and C. Polychropoulos, "A detailed, transistor-level energy model for SRAM-based caches", *Proc. Int. Sym. Circuits and Sytems*, 1999.

[18] N. Vijaykrishnan, M. Kandemir, M.J., Irwin, H.S. Kim and W. Ye, "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower", *Proc. of 27th Int. Sym. On Computer Architecture (ISCA)*, 2000.

[19] G. Reinman and N. Jouppi, "An Integrated Cache Timing and Power Model", *CACTI 2.0 Technical Report, COMPAQ Western Research Lab*, 1999.

[20] L. Gwennap, "Digital 21264 Sets New Standard", *Microprocessor Report*, vol. 10, no. 14, Oct. 1996.

[21] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors", *IEEE Journal of Solid-State Circuits*, 31(9), 1996.

[22] H-H. S. Lee, G. S. Tyson, and M. K. Farrens, "Eager Writeback – A Technique for Improving Bandwidth Utilization", MICRO 33, Dec. 2000.

[23] H. Zhou, M. Toburen, E. Rotenberg, T. Conte, "AMC: A low leakage power efficient on-chip cache system design", Technical Report, Department of Electrical and Computer Engineering, North Carolina State University, Nov. 2000.

[24] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: generational behavior to reduce cache leakage power", *Proc. of 28th Int. Symp. On Computer Architecture (ISCA)*, 2001.