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

HUIYANG ZHOU, MARK C. TOBUREN, ERIC ROTENBERG,
and THOMAS M. CONTE
North Carolina State University, Raleigh

Lower threshold voltages in deep submicron 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 that 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 64 KB caches.

Categories and Subject Descriptors: B.3 [**Hardware**]: Memory Structures—*design styles—cache memories*; C.1 [**Computer System Organization**]: Processor Architectures—*parallel architectures—mobile processors*

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Cache, static power, adaptive mode control

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 [1999] 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 a decrease in threshold voltage with each new generation.

Caches consume a significant fraction of total die area, especially in high-performance embedded processors, for example, 60% of the StrongARM die area is cache [Montanaro et al. 1997]. 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* [Yang et al. 2001] and *cache line decay* [Kaxiras et al. 2000]. Both approaches exploit a circuit technique called *gated-Vdd* [Powell et al. 2000], 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 reaccessed. However, caches trade-off 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 [Yang et al. 2001] 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 predetermined 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 reactivated to help reduce the observed miss rate.

Cache line decay [Kaxiras et al. 2000] 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 predetermined amount of time, and is reactivated only when it is reaccessed.

A limitation of both DRI and cache line decay is that 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 as 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 64 KB 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 do not always guarantee best results because (1) tuning reflects the behavior of profiled runs, whereas any given run may behave differently;
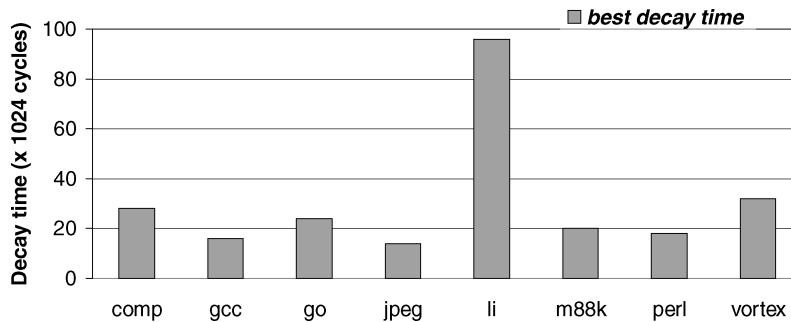
Fig. 1.   Per-benchmark cache line decay times, tuned to reduce performance by no more than 4%.

and (2) static parameters cannot capture variations within a single run of the program.

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. Predetermined 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 versus 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 from previous approaches is that the turn-off interval is dynamically adjusted to ensure performance that closely tracks the performance of an equivalent cache without sleep mode. That is, the turn-off interval is a 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, that is, 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.
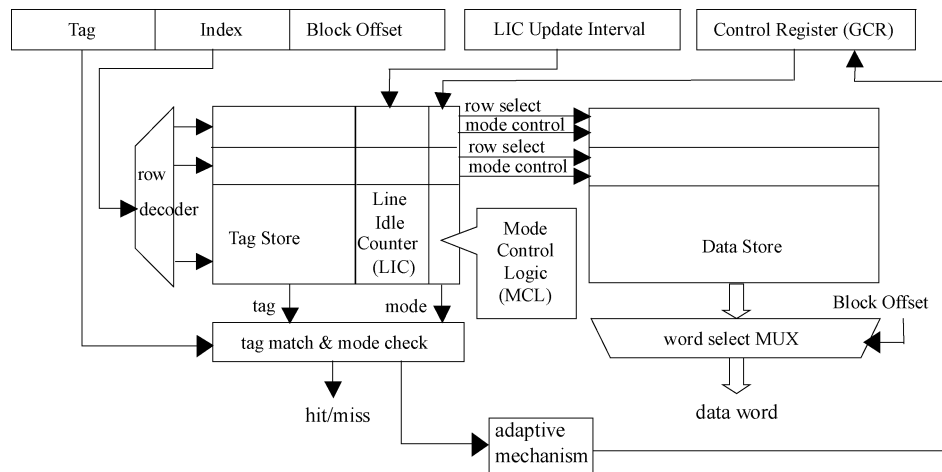
Fig. 2.   AMC cache architecture (direct mapped cache shown).

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 64 KB instruction and data caches.

This work uses the tag store to identify extra misses caused by sleep-mode lines, and adapts accordingly. Kaxiras et al. [2001] proposed an alternative adaptive technique based on the generational model of cache lines. A counter is associated with each cache line to predict whether a miss to a line is an ideal miss (start of a new generation) or a sleep miss (same generation). In Section 6 (Related Work), we discuss their approach further, and also compare power savings and performance of their approach with ours.

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 simulation methodology and results are presented in Sections 3 and 4, respectively. Energy savings is evaluated in Section 5. Section 6 discusses related work, and Section 7 concludes the paper.

2. ADAPTIVE MODE CONTROL

In this section, we first describe the overall AMC cache architecture (Section 2.1) and the adaptive mechanism (Section 2.2). Then, we discuss implications of keeping only a subset of tag bits active to reduce static power in the tag array. Finally, we briefly review the SRAM cell design with sleep-mode capability.

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).

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; this is called the *LIC update interval*. The LIC update interval is constant and implementation-dependent, the choice of which depends on a straightforward trade-off 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 [Kaxiras et al. 2000]. Detailed analysis in Sections 4.4 and 5 shows that a LIC update interval of 2048 cycles yields effective results with a small counter area overhead and negligible dynamic power overhead (LICs increment infrequently—once every 2048 cycles).

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 need to be written back to the L2 cache/main memory before a line is placed in sleep mode because data are 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 [Lee et al. 2000] that suggests it is possible for performance to improve. For write-through data caches, there is no need to write data to the L2 cache/main memory before deactivating the line, because the write-through policy ensures that the copy in the L2 cache/main memory is up-to-date.

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, that is, data must be fetched from the L2 cache and the cache line is reactivated 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
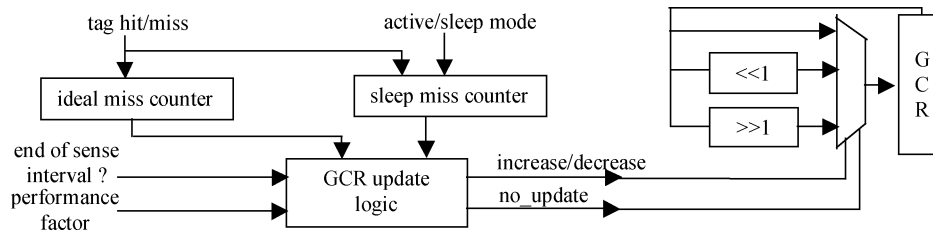
Fig. 3.   Adaptive mechanism for dynamically updating the turn-off interval stored in the GCR.

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.

The LIC counters and adaptive control logic consume power themselves and increase the area of the cache. However, these components consume dynamic power very intermittently, once every 2048 cycles and 1 million cycles, respectively. Therefore, our energy calculations in Section 5 show that the additional dynamic power is quite small. Static power due to LIC counters is more significant and is calculated in Section 5. As a final point, note that the AMC logic is not on the critical path of tag checking and data access. So AMC does not increase the hit time of the cache.

2.2 Adaptive Mechanism

The fact that cache tags are never put into sleep mode allows hardware to classify misses as ideal misses (misses that would have occurred regardless of a line's sleep/active status) or sleep misses (extra misses caused by sleep-mode lines). (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 and 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 4.4 that the duration of the sense interval has little impact on AMC results (both performance and power savings). The performance factor is set according to the desired balance between performance degradation and static power savings, and it can be controlled using a software-addressable register.

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)
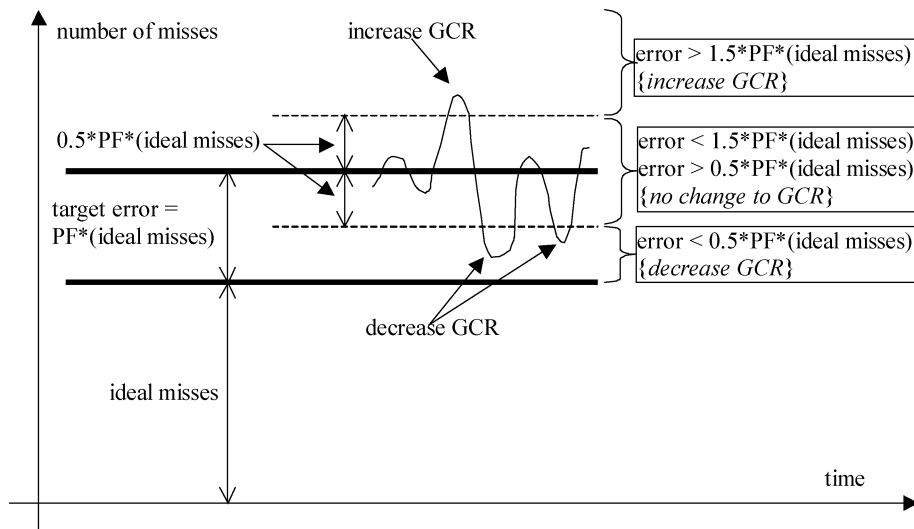{*decrease GCR*}

decrease GCR

ideal misses

time

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

   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 zero 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 the base quantity—in this case the number of ideal misses (such as 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

```
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
}
```

Fig. 5.   The GCR update algorithm.

misses) $\pm$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 1 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 1 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 example, in the 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.

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, $\frac{1}{2}$, $\frac{1}{4}$ and so on. 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 [Zhou et al. 2000].

It should be noted that any control system necessarily has predefined constants, PF in the case of AMC, miss bound and size bound in the case of DRI [Yang et al. 2001], and decay time in the case of cache line decay [Kaxiras et al. 2000]. All of these schemes dynamically track performance. AMC's distinction is that it tracks a dynamic and accurate performance target (ideal misses) instead of a predefined and potentially less accurate performance target. PF, albeit a
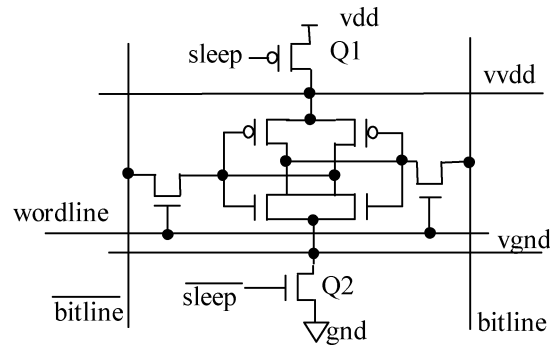
Fig. 6.   SRAM cell with sleep-mode support.

predefined parameter, is a multiplicative coefficient and determines how *closely* AMC performance tracks hypothetical performance.

2.3 Using Partial Tags

The method discussed in Section 2.2 keeps the full tag array active to precisely classify misses as either ideal misses or sleep misses. A disadvantage of keeping tags active is that the tags always consume static power.

Static power consumption in the tag array can be reduced by keeping only a part of the tag active when the corresponding cache line is in sleep mode, in exchange for less precise miss classification. When a deactivated line is accessed, only the tag bits that are active are checked to determine whether the miss is an ideal miss or a sleep miss.

If complete tags are used, a tag match indicates that a miss is definitely a sleep miss. However, if partial tags are used, a tag match indicates that a miss *may* be a sleep miss. AMC must conservatively classify it as a sleep miss. Because some ideal misses are misclassified as sleep misses, the measured sleep miss rate is larger than it actually is, and AMC overcompensates by turning off fewer cache lines. So using partial tags trades static power savings in the data array for static power savings in the tag array. This trade-off is investigated in Section 4.6.

2.4 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 [Kuroda et al. 1996; Margala 1999; Nii et al. 1998; Powell et al. 2000; Shigematsu et al. 1997]. 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 [Ye et al. 1998].

Table I. 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

Several variants of this implementation and the implications on power, area, and performance are discussed in Powell et al. [2000].

The cell can be improved by placing diodes between vgnd-gnd and vvdd-vdd, retaining data in sleep mode [Nii et al. 1998]. There is still a performance penalty for accessing a line that is in sleep mode, because it takes some time to turn the line back on. Therefore, AMC is still needed to moderate the performance/power trade-off. The advantage of the new cell, compared to one that does not retain data, is that turning the cell back on does not require refetching data from the L2 cache. Hence, the dynamic power cost of reaccessing sleep-mode lines is eliminated.

3. METHODOLOGY

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

3.1 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 [Burger and Austin 1997]. The underlying processor organization is based on the MIPS R10000 processor, configured as indicated in Table I. The SPECint95 benchmarks, listed in Table II, were run to completion.

3.2 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 $\frac{1}{2}$, 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 4.4, we examine the

Table II. Benchmarks

Benchmark	Input Dataset	Instruction Count (in million)
compress	compress95.ref	24
gcc	−O3 genrecog.i −o genrecog.s	117
go	99	133
jpeg	vigo.ppm	166
li	test.lsp (queens 7)	202
m88ksim	−c < ctl.in (dcrand.big)	120
perl	scrabble.pl < scrabble.in (dictionary)	108
vortex	vortex.in (persons.250, bendian.*)	101

impact of varying each of these parameters individually on static power and performance. The lower and upper bounds of the turn-off interval stored in the GCR are 4k and 128k cycles, respectively.

4. 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.

4.1 AMC Instruction Caches

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

Figure 7 shows % IPC degradation over all benchmarks and I-cache configurations. Figure 8 shows the corresponding miss rates, broken down into the sleep miss rate and ideal miss rate. The primary result from Figure 7, 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. First, 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, an absolute bound on the number of sleep misses. In Section 4.4, we study the effects of varying PF on both static power and performance.

Second, with 4-way set-associative caches, increasing cache size results in less performance degradation (except for a small deviation in *gcc* for the 32 kB 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.
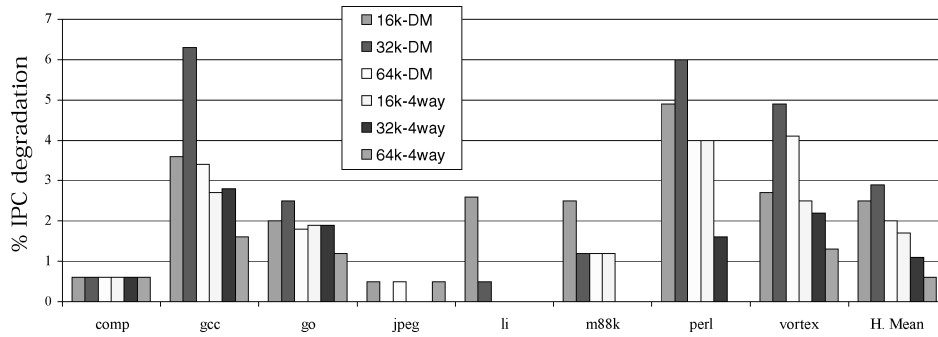
Fig. 7.   AMC I-cache: Performance degradation. (Bars that are not visible represent very low IPC degradation.)
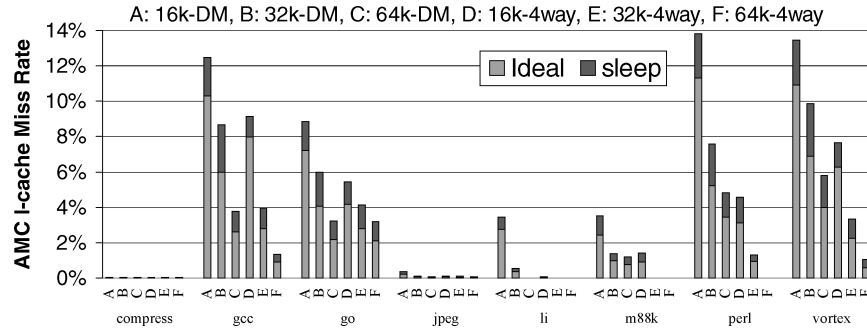

Fig. 8.   The breakdown of ideal miss rate and sleep miss rate for AMC I-caches. (Bars that are not visible represent very low miss rates.)

Third, 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 16 kB direct-mapped cache is lower than with a 32 kB direct-mapped cache, yet we would expect it to be higher. The 16 kB cache has a higher ideal miss rate than the 32 kB cache. For *gcc*, Figure 8 reveals ideal miss rates of 10% and 6% for 16 kB and 32 kB, respectively. The adaptive mechanism should be more aggressive at turning off lines in the 16 kB cache than in the 32 kB cache, because the mechanism targets a sleep miss rate equal to PF* ideal miss rate, where PF $= \frac{1}{2}$. Figure 8 shows that the 32 kB sleep miss rate is 2.7%, which is about half of its ideal miss rate. However, the 16 kB sleep miss rate is 2.2%, only one-fifth of its ideal miss rate. The reason is that a lower bound on the turn-off interval (4K cycles) prevents the adaptive mechanism from being more aggressive at turning off lines. If the turn-off interval could dip below 4K cycles, then the adaptive mechanism could achieve its target sleep miss rate. In that case, the IPC degradation and turn-off ratio would increase for 16 kB. But it is encouraging that the lower bound on the turn-off interval also bounds the sleep miss rate when the ideal miss rate becomes too high (a built-in safety mechanism).

Interestingly, for *gcc* (our largest benchmark), the 16 kB 4-way set-associative cache shows slightly the same trend as the 16 kB direct-mapped
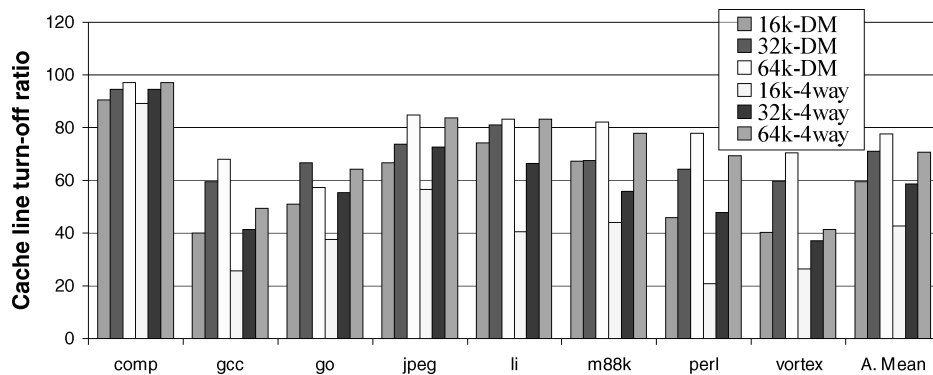
Fig. 9.   AMC I-cache: turn-off ratio.

cache. *Gcc* has the highest ideal miss rate among all benchmarks for this cache configuration. Therefore, its target sleep miss rate is also the most difficult to reach, given the lower bound on the turn-off interval. Once again, the minimum turn-off interval prevents the adaptive mechanism from achieving its target sleep miss rate.

From Figure 8, it can be seen that the sleep miss rate tracks the ideal miss rate (i.e., sleep miss rate is close to PF * ideal miss rate) for most cache configurations. This is not always true for the 16 kB direct-mapped cache. As discussed above, this is due to the lower bound on the turn-off interval.

Figure 9 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% (16 kB 4-way cache) to 77% (64 kB direct-mapped cache) turned-off cache-lines, justifying the relatively small performance degradation.

Two other trends are evident from Figure 9. 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 32 kB 4-way cache is effectively larger than a 32 kB direct-mapped cache, it does not mean the turn-off ratio will be higher for the associative cache. In fact, the *utilization* of the 32 kB 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 9 it is apparent that AMC is able to dynamically adjust to different behaviors among benchmarks. Specifically, the turn-off ratio varies substantially, while performance degradation is kept fairly low across all benchmarks. Turn-off ratios for a 32 kB 4-way set-associative instruction cache
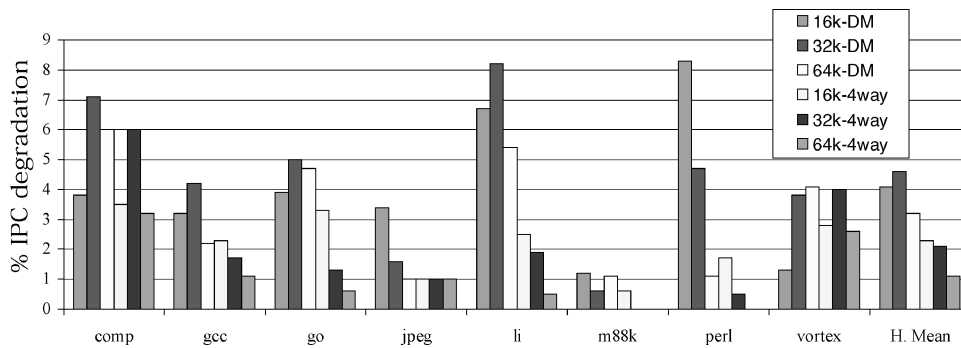
Fig. 10. AMC D-cache: Performance degradation. (Bars that are not visible represent very low IPC degradation.)

are 94% (*compress*), 41% (*gcc*), 55% (*go*), 73% (*jpeg*), 66% (*li*), 56% (*m88ksim*), 48% (*perl*), and 37% (*vortex*).

4.2 AMC Data Caches

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

Figure 10 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, and hence its sleep miss rates are higher.

From Figure 10, 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 16 kB caches. That is, for *compress*, *gcc*, *go*, *li*, and *vortex* with a 16 kB direct-mapped cache, 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 16 kB 4-way set-associative cache. (We even see the same trend for a 32 kB direct-mapped cache in *vortex*.)

Figure 11 shows the miss rates for AMC D-caches, broken down into the sleep miss rate and ideal miss rate. It can be seen that the actual sleep miss rate is close to the target sleep miss rate (PF* ideal miss rate), except for the 16 kB direct-mapped cache. The reason is the same as discussed in Section 4.1.

Figure 12 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% (16 kB 4-way cache) to 75% (64 kB direct-mapped cache).

Again, similar to AMC I-caches, we see from the average results in Figure 12 that AMC D-cache turn-off ratios tend to decrease with decreasing cache size and decrease with increasing associativity. However, for D-caches,

A: 16k-DM, B: 32k-DM, C: 64k-DM, D: 16k-4way, E: 32k-4way, F: 64k-4way
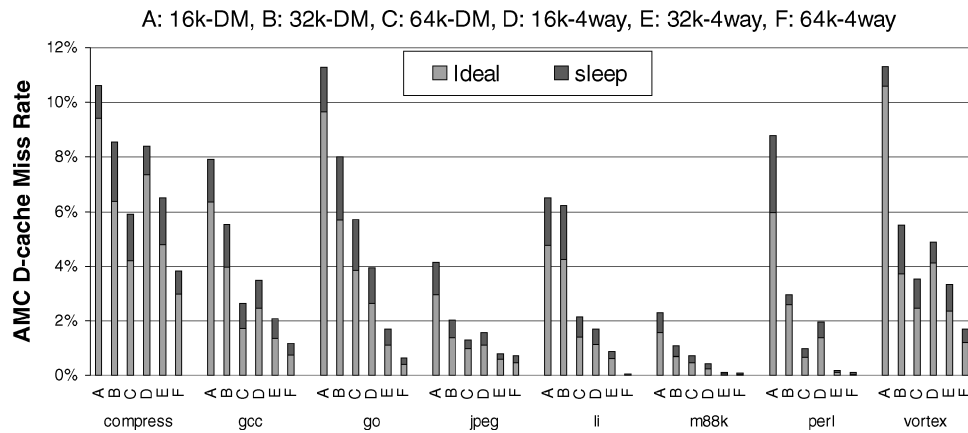

Fig. 11.   The breakdown of ideal miss rate and sleep miss rate for AMC D-caches. (Bars that are not visible represent very low miss rates.)
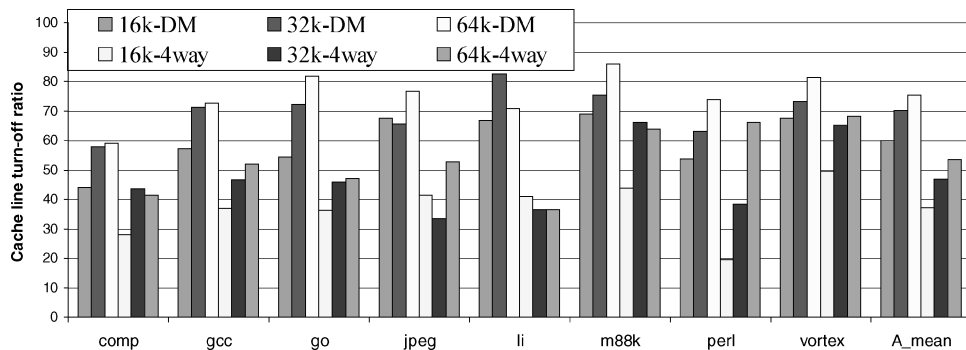

Fig. 12.   AMC D-cache: Turn-off ratio.

there are more deviations from these average trends among individual benchmarks.

4.3 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 64 kB 2-way I-cache and 64 kB 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 4.1 and 4.2. Note that $PF = \frac{1}{2}$ for both D-cache and I-cache, and each has a separate GCR.

4.4 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

Table III. Simulation Results with Different Performance Factors (PFs)

	IPC				I-cache turn-off ratio (%)			D-cache turn-off ratio (%)		
Benchmark	base (no AMC)	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	1.87	1.86	1.85	1.82	64.2	67.5	76.2	46.4	49.6	60.3

phases is ideal, our studies do not show a significant difference when the sense interval is varied from 250 000 to 4 000 000 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 III shows results for each benchmark using a 64 kB 2-way I-cache and 64 kB 4-way D-cache, in which we varied PF from 1/8 to 1 (again, PF is the same for both caches 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 of each benchmark showed maximum 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 that a 2048-cycle interval provides a good trade-off between counter overhead (area and dynamic power) and aggressiveness in deactivating cache lines.

4.5 Comparison with Oracle Model

In this section, we compare AMC results with an oracle model of cache energy savings in the literature [Kaxiras et al. 2001]. In the oracle model, a cache line is deactivated for the interval between the final access to the line and the eviction of the line, called the *dead time*. This maximizes energy savings within the constraint of not introducing additional cache misses, that is, there is no performance degradation. In our experiments, a 64 KB 2-way set-associative
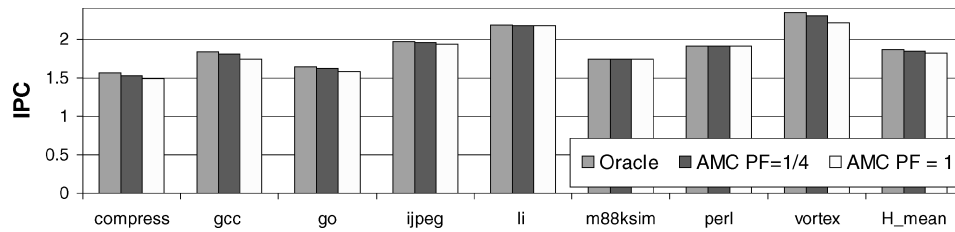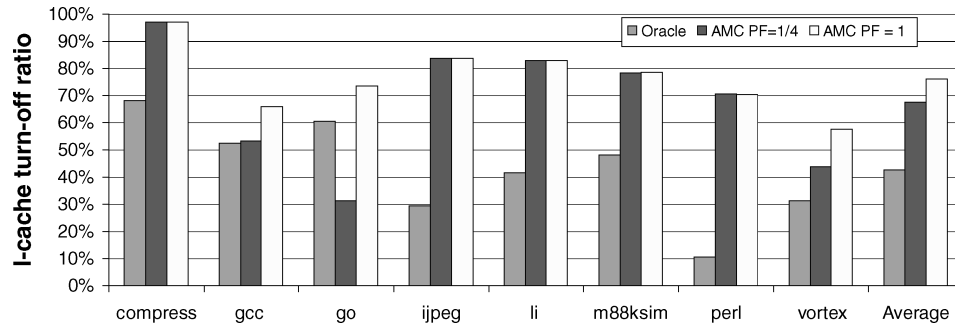
Fig. 13.   Performance of oracle and AMC.

Fig. 14.   I-cache turn-off ratios for oracle and AMC.

I-cache and a 64 KB 4-way set-associative D-cache are used. Figures 13, 14, and 15 show performance, I-cache turn-off ratio, and D-cache turn-off ratios, respectively. Results are shown for Oracle and for AMC with two different performance factors ($\frac{1}{4}$ and 1).

From Figure 13, AMC with PF $= \frac{1}{4}$ performs only 0% (*m88ksim, perl*) to 2.1% (*vortex*) worse than Oracle, and only 1.1% worse on average. AMC with PF $= 1$ performs only 0% (*m88ksim, perl*) to 6% (*vortex*) worse than Oracle, and only 2.7% worse on average.

From the I-cache turn-off ratios in Figure 14, the AMC I-cache with PF $= \frac{1}{4}$ achieves a much larger turn-off ratio than oracle for five out of eight benchmarks (*compress*, *ijpeg*, *li*, *m88ksim*, and *perl*), with at most only a 2% degradation in IPC. The reason is that AMC exploits long intervals between accesses to the same cache line, without introducing too many sleep misses. Oracle does not capture these long intervals because it does not permit any sleep misses. For *gcc* and *vortex*, the AMC I-cache with PF $= \frac{1}{4}$ achieves slightly larger to moderately larger turn-off ratios than oracle, with at most only a 2.1% degradation in IPC. For *go*, the AMC I-cache with PF $= \frac{1}{4}$ fails to exploit the short dead time between the final access to a line and the eviction of the line. When PF is increased to 1, turn-off intervals are smaller and those opportunities are utilized.

From Figure 15, AMC with PF $= \frac{1}{4}$ gives lower D-cache turn-off ratios than oracle for six out of eight benchmarks, unlike the I-cache trend in Figure 14. The reason is the D-cache has shorter dead times than the I-cache. AMC with PF $= \frac{1}{4}$ fails to capture the shorter dead times whereas oracle does capture these. Using PF $= 1$ allows AMC to capture the shorter dead times, too.
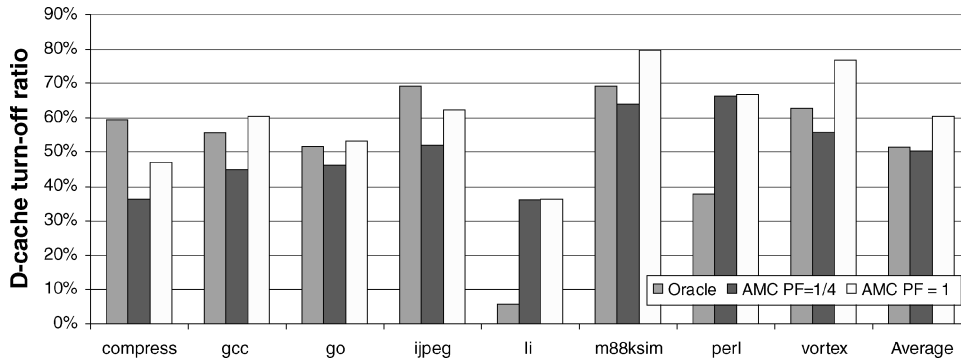
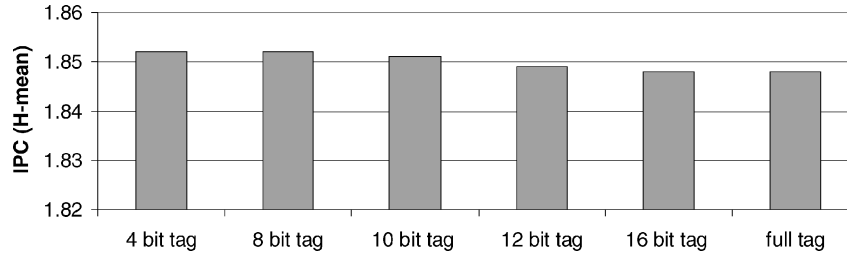Fig. 15.   D-cache turn-off ratios for oracle and AMC.

Fig. 16.   Performance as the number of active tag bits is varied.

In summary, compared to oracle, AMC with PF $= \frac{1}{4}$ achieves a significantly higher I-cache turn-off ratio (68%, compared to oracle's 43%) and a comparable D-cache turn-off ratio (50%, compared to oracle's 51%) with only a 1.1% performance degradation, on average.

4.6 Impact of Using Partial Tag Bits

As discussed in Section 2.3, the static power consumed by the tag array can be reduced by keeping only part of the tag active when the corresponding cache line is deactivated. In our experiments, we vary the number of active tag bits from 4 bits to the full tag. A 64 KB 2-way set-associative I-cache and a 64 KB 4-way set-associative D-cache are used and PF is fixed at $\frac{1}{4}$.

Results are averaged over all benchmarks. Figure 16 shows performance. Figure 17 shows the *data array turn-off ratio*, which is the fraction of bits in the data array that are turned off. Note that this ratio is equivalent to the cache line turn-off ratio used in previous sections (because all data bits in a cache line are turned off). Figure 18 shows the *tag array turn-off ratio*, which is the fraction of bits in the tag array that are turned off. Figure 19 shows the *overall turn-off ratio*, which is the fraction of bits in the combined tag and data arrays that are turned off.

Figures 16 and 17 show that using fewer tag bits results in lower data array turn-off ratios and higher performance. When the number of tag bits is decreased from the full tag to 4 bits, performance increases by 0.004 IPC. The change in data array turn-off ratio is more noticeable. The I-cache turn-off ratio
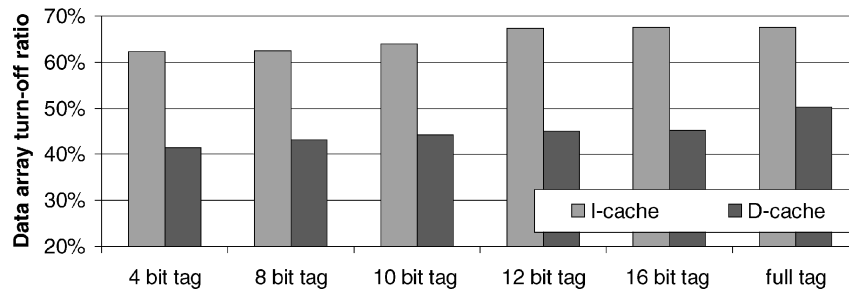
Fig. 17.   I-cache and D-cache data array turn-off ratio as the number of active tag bits is varied.
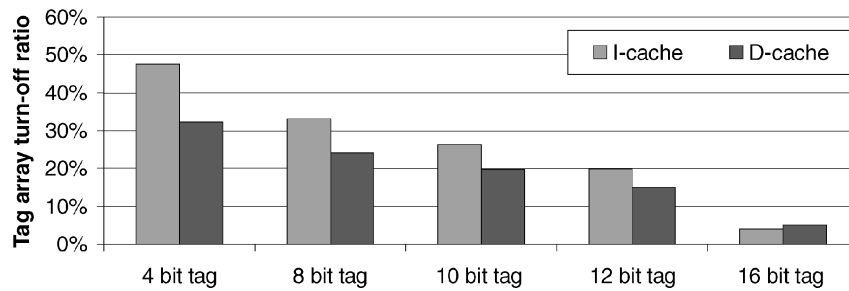

Fig. 18.   I-cache and D-cache tag array turn-off ratio as the number of active tag bits is varied.
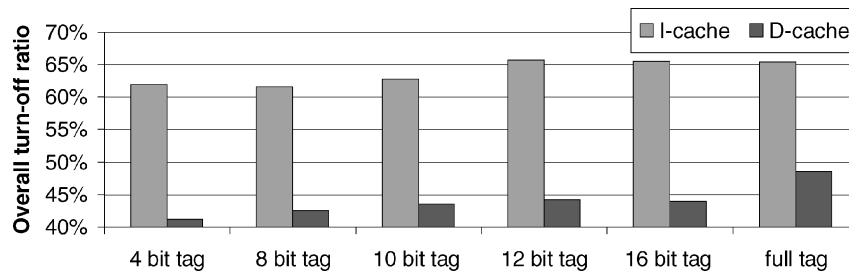
Fig. 19.   I-cache and D-cache overall turn-off ratio as the number of active tag bits is varied.

decreases from 67% to 62%, and the D-cache turn-off ratio decreases from 50% to 42%. AMC tries to maintain a sleep miss rate that is a factor PF of the ideal miss rate (sleep_misses = PF*ideal_misses). As discussed in Section 2.3, using fewer tag bits misclassifies some ideal misses as sleep misses. Therefore, the number of ideal misses is underestimated, and AMC tries to reduce the number of sleep misses accordingly by increasing the turn-off interval and deactivating fewer cache lines.

Figure 18 shows that the tag array turn-off ratio increases with fewer active tag bits, as expected. When the number of active tag bits is reduced from 16 bits to 4 bits, the I-cache tag array turn-off ratio increases from 4% to 48% and the D-cache tag array turn-off ratio increases from 5% to 32%.

From Figures 17 and 18, using partial tags trades power savings in the data array for power savings in the tag array. The net effect is quantified in Figure 19,

which shows the overall turn-off ratio. For the I-cache, there is a slight increase in overall static power savings if 12 bits are used instead of the full tag. However, the improvement is minimal. For the D-cache, using fewer than the full number of tag bits always reduces the overall static power savings. The reason is the static power consumed by the tag array is very small (3.3%) compared to the static power consumed by the data array, in our cache configuration. The number of tag bits for the I-cache and D-cache is 17 and 18, respectively, and the number of bits in the data portion of the line is 512.

In summary, while using partial tags saves static power in the tag array, it reduces static power savings in the data array and the net effect does not validate its effectiveness. The conclusion may differ for smaller cache line sizes, where the number of tag bits is a significant fraction of the cache line.

5. ENERGY ANALYSIS

In this section, we evaluate the overall energy savings of AMC, taking into account the energy overhead of the additional AMC logic. We base our analysis on experiments using the Compaq 21264 I-cache, a 64 KB 2-way set-associative cache implemented in a 0.35 μm CMOS process. In Section 5.1, we discuss the input parameters to our analysis and calculate the energy overhead of the LIC and MCL logic. In Sections 5.2 and 5.3, we determine the total energy savings of AMC at the cache level and the processor level, respectively. We find that AMC, implemented in a 64 KB 2-way set-associative I-cache using a 0.35 μm process, yields energy savings of 6.4% at the cache level and 2.0% at the processor level. The savings are expected to be much higher in future deep submicron technologies.

5.1 Parameters Used in the Energy Analysis

In order to determine the overall energy savings of AMC, we need values for the parameters listed in Table IV. For each parameter, Table IV gives the parameter's name, a brief description, its value, and the method used to measure the value.

The static energy consumption of the cache data array was calculated from the 0.35 μm leakage current curves for n-type and p-type transistors as defined in the process specification and the device widths of the data array SRAM cells. This calculation, when factored by the data array dimensions, yielded a total static energy consumption of 0.17 nJ. To determine values for the next five parameters in the table, we performed a series of SPICE experiments using layout-extracted SRAM cells and 7-bit counters taken from the 21264 I-cache and control logic. We first measured the dynamic energy consumed in a single data array read or write access by isolating the energy consumed in the bit-lines, wordlines, and within the SRAM cell itself. This resulted in a dynamic energy consumption of 0.16 nJ. Because the 21264 I-cache typically performs three transactions within a single cycle (two reads and one write), the static-to-dynamic energy ratio of the I-cache data array is 1:3. It is important to note here that this ratio will increase dramatically as process technology shrinks [Bellas et al. 1999]. The effects of this will be discussed later in this section.

Table IV. Parameters Used in the Energy Analysis

Parameter	Description	Value	Measurement Method
$E_{\text{da\_static}}$	Static energy consumption of the cache data array	0.17 nJ	Calculated from process parameters, cell layout, and cache dimensions
$E_{\text{da\_dynamic}}$	Dynamic energy consumption of the cache data array	0.16 nJ	SPICE
$R_{\text{da\_static\_dynamic}}$	Static-to-dynamic energy ratio within the cache data array	1:3	Typically two reads and one write per cycle: $E_{\text{da\_static}}/(3*E_{\text{da\_dynamic}})$
$E_{\text{LIC\_static}}$	Static energy consumed in the LIC/MCL logic	15.5 fJ/counter 15.9 pJ total	Calculated from process parameters, 21264 7-bit counter layout
$E_{\text{LIC\_dynamic}}$	Dynamic energy consumed in the LIC/MCL logic	~0.0 fJ	SPICE, approximation based on measured energy and LIC update interval
$E_{\text{L2\_sleep}}$	Dynamic energy consumed by L2 accesses resulting from sleep misses	1.6 pJ	SPICE (energy of L2 bank access approximately equal to $E_{\text{da\_dynamic}}$), and sleep miss rate
$P_{\text{da\_cache}}$	Percent of cache energy consumed by data array	40%	Literature survey [Reinman and Jouppi 1999]
$P_{\text{cache\_proc}}$	Percent of processor energy consumed by cache	30%	Literature survey [Bellas et al. 1999]

We used the 7-bit counter circuits taken from a random piece of the 21264 control logic to estimate the static and dynamic energy overhead of the LIC/MCL logic. A similar analysis using the device leakage characteristics and sizes was performed to determine the static portion of the energy consumption. Since the LIC/MCL logic is relatively small in comparison to the size of the data array of the cache, its corresponding static energy consumption, measured at 15.5 fJ, is also relatively small. However, the static energy consumption is 15.9 pJ for all LICs combined, which is about 10% of the static energy consumed by the cache's data array. Since the LICs are updated only every 2048 cycles, their dynamic energy is negligible in comparison to the dynamic energy consumed in a single data array access. Therefore, we assume a value of 0.0 fJ for the dynamic energy consumption of the LIC/MCL logic.

We used the SPICE experiments on the L1 data array circuits to determine the energy overhead due to additional L2 accesses resulting from L1 sleep misses. This overhead is primarily a function of the sleep miss rate and is determined by measuring the dynamic energy cost of a single L2 access and multiplying by the sleep miss rate. The 21264 L2 cache is structured, from a circuit design perspective, very similarly to the L1 caches. The cache bank structure, which determines the energy consumption of a data array access, is approximately the same from a circuit design perspective (width and depth) for both caches. Therefore, we used the SPICE measurement of the L1 data array, 0.16 nJ, to estimate the additional dynamic energy of accessing a single L2 bank as a result of a sleep miss. The architectural analysis of AMC using a 64 KB 2-way set-associative I-cache gives a maximum sleep miss

rate across all benchmarks of 1%. This yields a total L2 access overhead of 1.6 pJ.

Finally, we used data taken from the literature to estimate the contribution of data array energy consumption to total cache consumption [Reinman and Jouppi 1999] and the contribution of total cache consumption to processor consumption [Bellas et al. 1999]. In developing the CACTI cache analysis tool, Reinman and Jouppi [1999] found that the data array is responsible for approximately 40% of the total cache energy. Bellas et al. [1999] estimate that the cache contributes from 20 to 40% of the total processor energy consumption. In our analysis, we use values of 40% and 30% for $P_{\text{da\_cache}}$ and $P_{\text{cache\_proc}}$, respectively.

5.2 Cache-Level Energy Savings

In this section, we use the parameters from Section 5.1 to determine the total cache energy savings of AMC based on the 0.35 μm process technology. We must first determine the total static energy savings in the data array of the cache, given by:

$$\%_{\text{da\_static\_save}} = \frac{(E_{\text{da\_static}} \times \text{turnoff\_ratio}) - (E_{\text{LIC\_static}} + E_{\text{LIC\_dynamic}} + E_{\text{L2\_sleep}})}{E_{\text{da\_static}}} \times 100\%.$$

As indicated in Section 5.1, due to the low sleep miss rate and infrequent LIC updates, most of the energy overhead of AMC is the static energy consumed by the LICs ($E_{\text{LIC\_static}}$). Using the parameters in Table IV, the energy overhead is about 10% of the static energy consumption in the data array. So the total static energy savings in the data array is the turn-off ratio minus 10%. As the chosen AMC I-cache achieves a 73% turn-off ratio (see Section 4.3), the total static energy savings in the data array is 63%. Applying the static-to-dynamic energy ratio, $R_{\text{da\_static\_dynamic}}$, of 1:3, we find that the total energy savings of the data array is 16%. Factoring this number by $P_{\text{da\_cache}}$, we find that the total cache energy savings using AMC, assuming 0.35 μm process technology, is 6.4%.

5.3 Processor-Level Energy Savings

In this section, we extend the energy savings analysis to determine the overall savings at the processor level. As previously stated, in general, caches consume anywhere from 20 to 40% of the total processor energy [Bellas et al. 1999]. Based on these findings, we assume a $P_{\text{cache\_proc}}$ value of 30%. Combining this assumption with the results from the previous section, we find that AMC results in a total energy savings of 2.0% for the entire processor.

At this point, it is important to note how the static-to-dynamic energy ratio, $R_{\text{da\_static\_dynamic}}$, affects both the cache-level and processor-level energy savings. This ratio increases dramatically as process size shrinks, and it is predicted that static energy consumption may become equal to dynamic energy consumption in as few as two process generations [Butts and Sohi 2000]. As a result, the total static energy savings of AMC will increase with process shrinks and provide greater overall energy savings at the cache and processor levels.

Table V. The Energy Savings for Different Deep Submicron Technologies (Projections Based on Constant-Electric Field Scaling)

	0.35 μm	0.25 μm	0.18 μm	0.13 μm
Percent savings in data array static energy	62.7%	63.5%	63.6%	63.6%
Percent energy saved in total data array	16.4%	45.2%	60.1%	63.1%
Percent energy saved in total cache	6.6%	18.1%	24.5%	25.2%
Percent energy saved in total processor	2.0%	5.4%	7.2%	7.6%

Table V shows projected energy savings for smaller feature sizes. Our projections are based on scaling factors derived by Borkar [1999]. Assuming constant electric field scaling and a fixed number of transistors, static power consumption increases by a factor of 3.5 and dynamic power consumption decreases by a factor of 0.5. The energy savings in Table V show that the effectiveness of AMC increases substantially for deep submicron technologies.

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 [Albonesi 1999] 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 [Bellas et al. 2000] and filter cache [Kin et al. 1997] 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 [Kamble and Ghose 1997] 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. Subbanking in the data array [Kamble and Ghose 1997] and multiple divided modules (MDM) [Ko and Balsara 1995] also reduce the power consumption by accessing only part of the cache line. In addition to these techniques, several analytical energy models [Kamble and Ghose 1997; Reinman and Jouppi 1999; Vijaykrishnan et al. 2000] 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 [Yang et al. 2001], 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 [Powell et al. 2000]. As the I-cache size changes over time, an index remapping 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 pretuned for different applications. Cache line decay [Kaxiras et al. 2000] 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 predefined 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 [Kaxiras et al. 2001]. This approach associates a two-bit state machine with each cache line to gauge generational behavior. After a cache line is turned off, if the next access to the line occurs before $\frac{1}{4}$ of the turn-off interval, the access is assumed to be from the same generation. This implies the cache line decayed too early. So the turn-off interval is increased. If the next access to the decayed cache line occurs after $\frac{3}{4}$ of the turn-off interval, the access is assumed to be from a different generation and the turn-off interval is decreased. Using the terms in this paper, the first scenario is a predicted-sleep miss and the second scenario is a predicted-ideal miss. Our implementation of this scheme, using their parameters [Kaxiras et al. 2001], shows that it is very aggressive in turning off cache lines. Our results confirm their observation that most live access intervals (i.e., the time between two accesses in the same generation) are short (less than 1k cycles) [Kaxiras et al. 2001]. Our measurements indicate that this trend holds for different cache configurations (e.g., for compress, 97% and 94% of live access intervals are less than 1k cycles for 16 kB direct-mapped and 64 kB 4-way D-caches, respectively). As a result, the turn-off interval gauged by their scheme is close to the lower bound of 1K cycles for many cache lines independent of cache configuration. For small caches, their scheme produces similar turn-off ratios (66% compared to 60% for AMC, for 16 kB direct-mapped) and similar performance degradation (5.5% compared to 4.1% for AMC, for 16 kB direct mapped), with the advantage of also deactivating the tag store. For the 64 kB 4-way D-cache, their scheme results in a much higher turn-off ratio (92% compared to 54% for AMC) at the cost of higher performance degradation (15% compared to 1% for AMC). AMC, on the other hand, has the ability to adjust the trade-off between static power savings and performance degradation using the performance factor (PF).

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 that higher associativity results in lower cache turn-off ratios. This was initially

counterintuitive, but the explanation is that 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.
- A comparison with "oracle," which maximizes energy savings within the constraint of zero performance degradation, reaffirms AMC's effectiveness.
- We showed that turning off some of the tag bits for sleep-mode lines actually reduces overall static power savings. Using fewer tag bits causes the number of perceived sleep misses to increase, and AMC compensates by deactivating fewer cache lines. We used a large line size, which means it is better to keep the full tag active than sacrifice power savings in the data array. The conclusion may change for smaller cache line sizes.
- We performed a detailed energy analysis based on SPICE simulations, layout extractions, process parameters, and other information taken from the Alpha 21264 instruction cache. AMC energy overheads were included in the analysis. The analysis calculated energy savings of 6.4% at the cache level and 2.0% at the processor level. The savings are expected to be much higher in future deep submicron technologies.

ACKNOWLEDGMENTS

REFERENCES

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

BELLAS, N., HAJJ, I., AND POLYCHROPOULOS, C. 1999. A detailed, transistor-level energy model for SRAM-based caches. In *Proceedings of International Symposium on Circuits and Systems*, vol. 6, 198–201.

BELLAS, N., HAJJ, I., POLYCHROPOULOS, C, AND STAMOULIS, G. 2000. Architectural and compiler techniques for energy reduction in high-performance microprocessors. *IEEE Transactions on VLSI Systems 8*, 3, 317–326.

BORKAR, S. 1999. Design challenges of technology scaling. *IEEE Micro 19*, 4 (July), 23–29.

BURGER, D. AND AUSTIN, T. M. 1997. *The Simplescalar Tool Set Version 2.0*. Tech. Rep., Computer Science Department, University of Wisconsin-Madison.

BUTTS, J. A. AND SOHI, G. S. 2000. A static power model for architects. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 33)* (December), 191–201.

GONZALEZ, R. AND HOROWITZ, M. 1996. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits 31*, 9, 1277–1284.

GWENNAP, L. 1996. Digital 21264 sets new standard. *Microprocessor Report 10*, 14 (October).

KAMBLE, M. B. AND GHOSE, K. 1997. Analytical energy dissipation models for low power caches. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)* (August), 143–148.

KAXIRAS, S., HU, Z., AND MARTONOSI, M. 2001. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of 28th International Symposium On Computer Architecture (ISCA)* (June), 240–251.

KAXIRAS, S., HU, Z., NARLIKAR, G., AND MCLELLAN, R. 2000. Cache-line decay: A mechanism to reduce cache leakage power. *IEEE Workshop on Power Aware Computer Systems* (November).

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. 1997. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 30)* (November), 184–193.

KO, U. AND BALSARA, P. T. 1995. Characterization and design of A low-power, high-performance cache architecture. In *Proceedings of International Symposium on VLSI Technology, Systems, and Applications*, 235–238.

KURODA, T., FUJITA, T., MITA, S., NAGAMATU, T., YOSHIOKA, S., SANO, F., NORISHIMA, M., MUROTA, M., KAKO, M., KINUGAWA, M., KAKUMU, M., SAKURAI, T. 1996. A 0.9 V, 150 MHz 10 mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage scheme. *IEEE Journal of Solid-State Circuits 31*, 1770–1779.

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

MARGALA, M. 1999. Low power SRAM circuit design. In *IEEE International Workshop on Memory Technology, Design, and Testing*, 115–122.

MONTANARO, J., WITEK, R. T., ANNE, K., BLACK, A. J., COOPER, E. M., DOBBERPUHL, D. W., DONAHUE, P. M., ENO, J., HOEPPNER, G. W., KRUCKEMYER, D., LEE, T. H., LIN, P. C. M., MADDEN, L., MURRAY, D., PEARCE, M. H., SANTHANAM, S., SNYDER, K. J., STEPHANY, R., AND THIERAUF, S. C. 1997. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *Digital Technical Journal 9*, 49–62.

NII, K., MAKINO, H., TUJIHASHI, Y., MORISHIMA, C., HAYAKAWA, Y., NUNOGAMI, H., ARAKAWA, T., AND HAMANO, H. 1998. A Low power SRAM using auto-backgate-bontrolled MT—CMOS. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 293–298.

POWELL, M., YANG, S.-H., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. 2000. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 90–95.

REINMAN, G. AND JOUPPI, N. 1999. An itegrated cache timing and power model. *CACTI 2.0 Tech. Rep.*, COMPAQ Western Research Lab.

SHIGEMATSU, S., MUTOH, S., MATSUYA, Y., TANABE, Y., AND YAMADA, J. 1997. A 1-V high speed MTCMOS circuit scheme for power-down application circuits. *IEEE Journal of Solid-State Circuits 32*, 861–869.

VIJAYKRISHNAN, N., KANDEMIR, M., IRWIN, M. J., KIM, H. S., AND YE, W. 2000. Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower. In *Proceedings of 27th International Symposium on Computer Architecture (ISCA)* (June), 95–106.

YANG, S.-H., POWELL, M., FALSAFI, B., ROY, K. AND VIJAYKUMAR, T. 2001. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)* (January), 147–157.

YE, Y., BORKAR, S., AND DE, V. 1998. A new technique for standby leakage reduction in high performance circuits. In *IEEE Symposium on VLSI Circuits*, 40–41.

ZHOU, H., TOBUREN, M., ROTENBERG, E., AND CONTE, T. M. 2000. AMC: *A Low Leakage Power Efficient On-chip Cache System Design*. Tech. Rep., Department of Electrical and Computer Engineering, North Carolina State University.