Dynamic Leakage-Energy Management of Secondary Caches

Erik G. Hallnor and Steven K. Reinhardt

Advanced Computer Architecture Laboratory
EECS Department
University of Michigan
Ann Arbor, MI 48109-2122
{ehallnor,stever}@eecs.umich.edu

Static leakage currents are the prime contributor to energy consumption for large on-chip secondary/tertiary caches. This energy cost can be reduced by dynamically disabling unneeded portions of the cache. To provide overall energy savings, however, the leakage-energy reduction must exceed the extra energy incurred due to additional off-chip accesses and increased runtime. We derive a practical formula for on-line estimation of net energy reduction based on the number of additional misses incurred and the ratio of off-chip access energy to per-cycle cache leakage energy. We estimate reasonable values for this ratio by measuring the actual off-chip access energy cost on a current system. We incorporate our estimation formula into a previously proposed dynamic resizing scheme, preventing it from increasing net energy consumption and improving its overall effectiveness.

We also present a new resizing framework that tracks misses and controls power at the granularity of associative ways. Per-way miss counters enable simultaneous estimation of net energy consumption at all possible cache sizes, allowing direct identification of the minimum-energy configuration. This structure also greatly reduces hardware overhead compared to block-level resizing schemes. We call the combination of this framework with our estimation formula Energy Aware Bank-Level Resizing *(EnABLR). Simulations show that, across the entire SPEC CPU2000 suite, EnABLR reduces secondary-cache leakage energy by up to 80%, without increasing overall memory-system energy consumption by more than 1% in any situation. Average energy reductions range 14% to 29%, depending on off-chip access energy costs.*

1. Introduction

As threshold voltages scale downward along with transistor feature sizes, static power dissipation due to subthreshold leakage current is becoming an increasingly significant fraction of total power consumption for microprocessors. Total leakage power is estimated to increase fivefold with each process generation [5], equalling dynamic power within a few generations [7].

At the same time, growing transistor budgets have led to the incorporation of large on-chip secondary caches on mid-range to high-performance microprocessors, including those used in laptop and desktop PCs as well as server systems.[1] On-chip secondary caches serve a valuable purpose in avoiding off-chip memory accesses, which are very costly in terms of both energy and performance. Because these caches contain large numbers of transistors, of which only a small fraction switch in any given cycle, leakage may account for as much as 80% of secondary cache energy consumption [17]. As these caches constitute a large fraction of microprocessor die area

1. For simplicity of discussion, we assume a two-level on-chip cache hierarchy, and use the term *secondary* to refer to the on-chip cache furthest from the processor. For systems with three levels of on-chip cache, our discussion and results apply equally well to the tertiary cache.

(and an even larger fraction of the total transistor budget), secondary cache leakage energy is likely to be a significant factor in overall microprocessor energy consumption.

Leakage energy can be reduced by dynamically disabling unused portions of the cache, e.g., by gating V_{dd} [13] or raising the threshold voltage [12]. This “cache resizing” approach is largely complementary to circuit-level optimizations that seek to reduce leakage on enabled memory cells, such as high-V_T or dual-V_T memory designs [9]. Recently proposed techniques [17, 11, 19] have shown that, for many benchmarks, significant portions of primary (L1) instruction and data caches can be disabled without corresponding increases in miss rates or execution times.

Of course, disabling a portion of a cache reduces total system energy only if the reduction in leakage energy is greater than the energy consumed by the overheads incurred. These overheads include the energy expended in additional accesses to the next level of the memory hierarchy and the added energy due to increased runtime. If the energy and performance penalties for additional cache misses are relatively small, as in the case of an L1 cache backed by an on-chip secondary (L2) cache, the potential downside of cache disabling may be minimal. However, misses in on-chip secondary caches require off-chip accesses, which can be very costly in terms of both energy and performance. Overly aggressive disabling may thus lead to large increases in both energy consumption and runtime.

In this paper, we develop a secondary-cache resizing scheme which addresses this problem by seeking explicitly to minimize overall memory-system energy consumption. To this end, we first present a general formula for estimating net relative energy reduction due to cache resizing. We then simplify this formula to make it amenable to on-line calculation. The simplified formula relies on three parameters. One—the fraction of the cache disabled—can be measured directly. We can track the second parameter—the number of additional misses caused by disabling that fraction of the cache—by not disabling the tags corresponding to disabled data blocks, as has been proposed for other schemes [19]. We assume that the final parameter—the ratio between the energy consumption of an off-chip access to that of one cycle of cache leakage—is provided externally. Because this ratio may depend on factors external to the cache design, such as the system configuration and the operating temperature, systems should allow for this value to be varied dynamically. We estimate a reasonable range of values for this ratio by combining published leakage figures with our own measurements of off-chip access energy costs. We then apply this formula to the previously proposed adaptive mode control (AMC) scheme [19], causing it to adapt its control parameter (the decay threshold) with the goal of maximizing the estimated energy reduction. Unlike the original AMC algorithm, which uses a target relative miss rate to adapt its threshold, our enhanced version never increases net memory-system energy consumption, and is more effective overall.

Unfortunately, although the AMC framework allows calculation of the net energy reduction provided by the current decay threshold setting, it cannot estimate what the net energy reduction would have been using a different threshold. Our variant of AMC must iteratively adjust the threshold to find the minimum-energy setting. We present a *bank-level resizing (BLR)* control scheme which addresses this shortcoming by combining resizing based on associative ways [2, 18] with per-way hit counters [15]. This combination allows a control algorithm to calculate the value of any miss-based metric (such as net energy reduction) simultaneously at all possible resize points, not just the current setting. Thus the optimal resize point can be selected directly, without iteration. Bank-level resizing also has a much lower hardware overhead than block-level schemes such as cache decay [11] and AMC [19]. BLR achieves these benefits by restricting resizing to a much coarser granularity; however, our results show that an 8-way associative cache—well within the typical range for on-chip secondary caches—provides overall energy reductions competitive with finer-granularity schemes.

Finally, we integrate our energy-aware metric with BLR to create a novel adaptive resizing scheme, *energy-aware bank-level resizing (EnABLR)*, which attempts to minimize net energy consumption based on an externally specified estimate of the relative energy cost of off-chip accesses vs. cache leakage. Simulations show that we are able to reduce energy by an average of 14% to 29%, depending on the off-chip energy ratio, while incurring less than 3% increase in execution time. Perhaps more importantly, unlike energy-oblivious schemes, EnABLR almost never increases system energy consumption.

The rest of the paper is organized as follows. In the next section we discuss previous work in cache leakage-energy management. Section 3 describes our formula for calculating net energy savings due to secondary-cache resizing. We then derive an estimate for one of the formula's key parameters, the ratio of off-chip access to cache leakage energy. We apply this estimation formula to the AMC resizing framework, and demonstrate that it significantly outperforms the original energy-oblivious control strategy. In Section 4, we present our bank-level resizing (BLR) framework and its energy-aware control algorithm (EnABLR). We present simulation results for EnABLR in Section 5, and conclude in Section 6.

2. Previous Work

Previous work on cache resizing addresses two issues: circuit techniques for placing portions of the cache into a low-leakage state, and architectural techniques for determining what fraction of the cache, if any, should be placed in such a state. This work falls into the latter category. We include a brief discussion of the former category as background.

2.1. Circuit Techniques for Reducing SRAM Leakage

One approach to reducing cache leakage is to fabricate cache transistors with a higher threshold voltage (V_t) via doping. This technique incurs a performance penalty, in that the increased V_t causes these transistors to switch more slowly; this penalty can be reduced by using high-V_t transistors only in the less performance-critical paths of the SRAM cell, an approach known as dual-V_t [9]. However, higher V_t values reduce, but do not eliminate, leakage current. As supply voltages cross below 1.0V, transistors with V_t high enough to make leakage energy negligible may not switch at acceptable speeds [16]. Thus high-V_t fabrication provides at best a partial solution to leakage energy.

The gated-V_{dd} technique [13] places high-V_t transistors on the path between a standard SRAM cell and the power rails. When these transistors are active, the SRAM cell operates at full speed. When the transistors are placed in an inactive state, the SRAM cell is disconnected from power, effectively eliminating leakage current in the cell itself. If the V_{dd}-gating transistors have high V_t values, the total leakage current seen in this disconnected state is practically negligible. However, when placed in the inactive state, all data stored in the SRAM is lost.

An alternative technique, MTCMOS [12], raises V_t in the SRAM cell dynamically to reduce leakage currents. This feat is accomplished using a separate, higher voltage to bias the PFET wells in the low-power state, and raising the ground and V_{dd} levels by two diode-drop voltages. Since only the threshold voltages are affected, the data stays resident, but transistor switching speeds are greatly raised. Recently, Flautner et al. [8] proposed an alternate circuit which uses a fixed V_t but dynamically lowers V_{dd} to place SRAM cells in a state-preserving “drowsy” mode. Both these schemes involve greater hardware complexity than gated V_{dd}, such as the need to route a separate power supply across the cache. They are also somewhat less effective at reducing leakage energy per bit, as they do not completely disconnect the SRAM cell from the power source. In this paper, we focus on using gated V_{dd} to reduce leakage energy at the circuit level.

2.2. Architectural Techniques for Cache Resizing

Given a circuit-level method such as gated V_{dd} for reducing leakage energy dynamically, a higher-level policy must determine when and how to apply the method. We describe three such architectural techniques: the DRI cache [17], cache decay [11], and adaptive mode control (AMC) [19]. The techniques use gated V_{dd} to disable portions of the cache, adjusting the effective cache size dynamically to match application requirements. Other cache-resizing techniques (e.g., [2, 4, 18]) focus on optimizing cycle time and/or dynamic power rather than leakage energy.

The DRI cache design [17] is intended to minimize the leakage energy of L1 instruction caches. The DRI scheme uses a predetermined *miss bound* for its resizing decisions. If the number of misses in a given interval is less than the miss bound, a section of the cache is turned off; if greater, a section is turned on. The cache is resized by adjusting the number of active sets, which restricts the cache to be grown or shrunk by factors of two. In addition, resizing changes the set of address bits used for indexing, complicating the tag-matching logic. By focusing on instruction caches, the DRI cache avoids the need to write back dirty data when the cache is shrunk.

Cache decay [11] is a distributed scheme in which individual cache blocks determine autonomously whether to put themselves into a powered-off state. A periodic global signal increments counters associated with each cache block. A block's counter is reset when the block is accessed. If the counter reaches a predetermined threshold value without an intervening access (three in [11]), the block is powered down. In a writeback cache, a block containing dirty data must be written back before powering itself down. On a miss, a powered-down block (if any) is reactivated to hold the new data in preference to replacing an existing block. The base cache decay scheme assumes a fixed decay interval. The authors note that the decay interval can be set according to the energy cost of a miss to bound the worst-case energy consumption; we will present results from this policy in Section 5. The paper also presents a sophisticated adaptive scheme that allows blocks to select from global decay signals of various frequencies based on their access rates, and discusses compiler- and profiling-based control schemes as well.

Adaptive Mode Control (AMC) [19] extends cache decay by adding feedback-based control of the counter threshold value. To provide feedback, AMC does not turn off tag entries when the corresponding data blocks are powered down. An access that matches the tag entry of a disabled block is a "sleep miss", a miss that would not have occurred in the absence of resizing. AMC can thus measure the number of additional misses incurred due to the resizing policy. The system also counts the number of misses that did not match any tags (i.e., that would have occurred even in the absence of resizing). The ratio of these values gives the relative increase in misses over the measurement interval, or equivalently, the relative increase in miss rate. AMC adjusts the decay counter threshold value to attempt to keep the increase in relative miss rate within a predefined window (1.25 to 1.75 in [19]).

Hanson et al. [10] compare the energy-delay product of L1 and L2 caches using dual-V_t, gated V_{dd}, and MTCMOS, where the latter two are managed by the cache decay scheme. They conclude that dual-V_t offers the best energy-delay product for L2 caches. However, they consider the L2 miss energy cost to be only the energy required to drive an address off chip. This restriction results in an estimate of 0.9 nJ per L2 cache miss, roughly three orders of magnitude smaller than the system-level energy cost we measure in Section 3.2. They also use the non-adaptive, energy-oblivious cache decay scheme to control gated V_{dd} and MTCMOS. An updated comparison using our off-chip energy parameters and our energy-aware resizing scheme would be interesting, but is outside the scope of this paper.

Overall, these schemes target primarily L1 caches, where the penalty in both performance and energy for additional misses to an on-chip L2 is relatively low. In particular, the ratio of dynamic L2 access energy to per-cycle L1 leakage energy is estimated to be in the range of 4 to 10 (though results for a range of values up to 100 are presented in [11]). In this situation, a very large number of additional misses must be incurred before the energy benefits of disabling a portion of the cache are negated. For example, the DRI cache work [17] shows that the energy benefit of their scheme is largely insensitive to the miss bound, and that the performance impact of additional misses tends to bound the minimum L1 instruction cache size much more than the energy cost.

In contrast, we show in Section 3 that the ratio of off-chip access energy to per-cycle L2 leakage energy is likely to be much higher, in the range of 50 to 100. Under these circumstances, it is much more likely that overly aggressive cache disabling will lead to a net increase in system energy consumption. Our proposed resizing scheme, described in Section 4.1, avoids this problem by using a direct estimate of net energy savings as its optimization metric. For an improved metric to be effective, however, we must first put in place a control scheme that is capable of resizing a cache to track a target metric accurately. We propose one such scheme in the following section.

3. Calculating Energy Savings from Secondary Cache Resizing

This section presents a brief analysis of the energy savings due to secondary-cache resizing. First, we derive an equation for estimating net energy savings. The ratio of off-chip access energy to secondary-cache leakage energy is a key parameter in this equation. We then estimate values for this ratio using published leakage values combined with measured off-chip energy costs.

3.1. Calculating Net Energy Savings

There are two primary sources of energy overhead induced by turning off portions of a secondary cache. First, the reduction in effective cache size leads to additional off-chip accesses, each of which consumes dynamic energy. Second, these additional off-chip accesses increase program runtime. The additional execution cycles could otherwise be spent on a different task or in a low-power idle mode. Thus the additional runtime potentially incurs dynamic and static energy overheads across the entire system, including the processor core, L1 caches, memory, and I/O devices. Additional dynamic energy dissipation in the processor core and elsewhere can be significantly reduced through power-saving techniques such as clock gating. Static energy consumption in other components may be significant, but is difficult to estimate within the scope of this paper. Therefore we focus on only two sources of overhead: dynamic energy due to extra off-chip accesses, and leakage energy dissipated by the L2 cache itself during the extra execution cycles.

Given these limitations, we can estimate the total energy savings as the leakage energy saved by resizing, minus the energy costs of off-chip accesses and additional runtime, as follows:

$$\text{Energy savings} = E_L(1 - F_A)t - E_{OC}(M + W) - E_L F_A(t' - t) \tag{1}$$

where:
E_L = cache leakage energy per cycle
F_A = active fraction of cache
t = runtime with full cache in cycles
t' = runtime with resized cache in cycles
E_{OC} = dynamic energy of an off-chip access

M = no. of additional misses vs. full-cache run
W = no. of additional writebacks vs. full-cache run

Given that our goal is to maximize energy savings, and not to quantify the energy saved, we can normalize the energy savings relative to the per-cycle leakage energy (E_L). We can then simplify Equation 1 slightly, replacing the absolute energy terms E_L and E_{OC} with the ratio of off-chip energy to leakage energy, $R_{OL} = E_{OC}/E_L$. (R_{OL} for secondary caches corresponds to the *L2Access:leak* ratio used for primary caches in [8].) Dividing Equation 1 by E_L thus gives:

$$\text{Normalized energy savings} = (1 - F_A)t - R_{OL}(M + W) - F_A(t' - t) \quad (2)$$

Unfortunately, even this simplified equation is not directly useful as an on-line estimate. Determining the effect of cache misses on total runtime ($t' - t$) is extremely difficult, given the widely varying latency tolerance of loads both within and across benchmarks [14]. Measuring the number of additional writebacks caused by resizing (W) is also challenging; we must distinguish writebacks that would *not* have occurred in the full-size cache, i.e., a dirty block that is written back, reloaded, and modified again in an interval where it would have remained in the full-size cache. We therefore eliminate these terms from consideration; we will show in Section 5 that these terms constitute a small part of the total energy overhead in practice. These simplifications lead to our final equation, which provides and estimate of the normalized energy savings over a period of t cycles:

$$\text{Est. energy savings} = (1 - F_A)t - R_{OL}M \quad (3)$$

We control the active fraction of the cache (F_A), and can directly measure the number of additional misses (M) if we maintain tags for disabled blocks (as in [19]). The key unknown is R_{OL}, the ratio of off-chip energy to leakage energy. We assume that the value of R_{OL} is provided externally, e.g., written into a control register by boot firmware or the operating system. In the following section, we estimate a range of values for R_{OL}. We will use these values in our simulations in Section 5 to determine the effectiveness of our scheme.

3.2. Estimating the Off-chip to Leakage Energy Ratio, R_{OL}

To find an estimate for the value of R_{OL}, we need to determine both the per-cycle leakage energy for a cache of a given size and the energy consumed by an off-chip access. We obtain the former from prior work. We found no published estimates for the latter, so we derived this value experimentally by measuring the power consumption of an actual system.

Table 2 in [17] indicates that a single SRAM bit in a 0.18 µm process has a leakage energy of $1.74\text{x}10^{-6}$ nJ per cycle at a temperature of 110C. Thus the data portion of a 1MB cache has a leakage energy per cycle of roughly 15 nJ. Given variations in process technology, cache sizes, and cycle times, this value is merely a ballpark estimate, but is adequate for our purposes.

Since the energy cost of an off-chip access has not been widely studied, we designed and performed our own experiment to measure it. We developed a microbenchmark that executes a simple loop containing a single memory access to a dynamically selected element of a large array. By exploiting the differences in size and associativity between the primary and secondary caches on the machine under test, the microbenchmark guarantees that this access always misses in the primary cache, and can choose dynamically whether the access also misses in the secondary cache. Parameters to the microbenchmark select a reference pattern of n L2 misses followed m hits. This pattern is repeated until a minimum amount of time has elapsed.

We instrumented a PC motherboard by introducing 0.1 ohm resistors on the power supply lines to the motherboard and tracking the voltage drop across these resistors. We integrate this measured power over the running time of the microbenchmark to determine the total motherboard energy consumption. We model this energy consumption as an equation with three unknowns: static energy per unit time, dynamic energy per microbenchmark iteration, and dynamic energy per L2 miss. The total runtime, number of iterations, and number of L2 misses for each microbenchmark run are known values. By measuring consumption across a variety of microbenchmark parameters, we can solve for dynamic energy per L2 miss. Although we do not account for variables such as operating-system activity, we minimize their effects by executing the microbenchmark runs for macroscopic time intervals (30 seconds) and averaging across multiple runs. Thus, for example, the energy cost of timer interrupts should be reasonably stable across our runs, and will be factored into the static energy per second component of our equations.

Performing this experiment on a system with a ASUS CUSL2-C motherboard, with a 933Mhz Intel Pentium III CPU, an Intel 815E chipset, and 128MB of main memory, we measured off-chip memory access costs ranging from 0.8 to 1.4 μJ. The variations arose from different patterns of L2 misses, which induced different levels of operation overlap in the memory system. However, for a given miss pattern, the energy measurements were very stable across multiple runs.

It is difficult to extrapolate these measurements to off-chip energy values for future systems; in fact, we expect a wide range of values across contemporary systems, as memory systems vary widely from laptop to workstation to server designs. However, as bus signaling and printed-circuit technology changes more slowly than semiconductor process technology, and as we are merely attempting to establish a reasonable range of energy ratios, we will simply use the measured range of 0.8 to 1.4 μJ per access. Combining this with the estimated 15 nJ per cycle leakage energy of a 1MB cache, we get a rough estimate of 50 to 100 for R_{OL}. To bracket both approximations in our estimate and trends in future technologies, as well as operating temperature variations, we consider R_{OL} values ranging from 50 to 200, in 2x steps.

3.3. Adding Energy Awareness to AMC

In this section, we demonstrate the effectiveness of our on-line net energy estimation formula by applying it to a previously proposed adaptive cache resizing framework. We chose adaptive mode control (AMC) [19] (see Section 2.2) as our baseline. We also implemented the adaptive cache decay scheme proposed by Kaxiras et al. [11] in our simulator, but found that AMC was consistently more effective. AMC's sleep miss measurements also provide a better foundation for net energy estimation.

The original AMC control algorithm varies the cache decay threshold to keep the total number of misses within a particular ratio of the number of "ideal misses" (those that would have occurred even in a fully enabled cache). The authors suggest a target ratio of 1.5 (i.e., allowing cache disabling to increases the number of misses by 50%). They sample hardware miss counters every million cycles, and adjust the decay threshold whenever the measured ratio goes below 1.25 or above 1.75. This algorithm is not energy aware: as discussed in Section 3.1, net energy savings is a function of the number of additional misses incurred ("sleep misses" in AMC terminology) and the number of disabled blocks, and is independent of the number of ideal misses.

We make AMC energy aware by changing only the criteria used to vary the decay threshold value. At the end of each million-cycle interval, we use Equation 3 to estimate the net energy savings for that interval, based on the measured number of sleep misses and the current number of disabled blocks. (Because blocks are enabled and disabled continuously throughout each interval, the number of disabled blocks at the end of the interval serves as a simple estimate of the average number disabled at any point during the interval.) Based on this estimate, we use a simple hill-

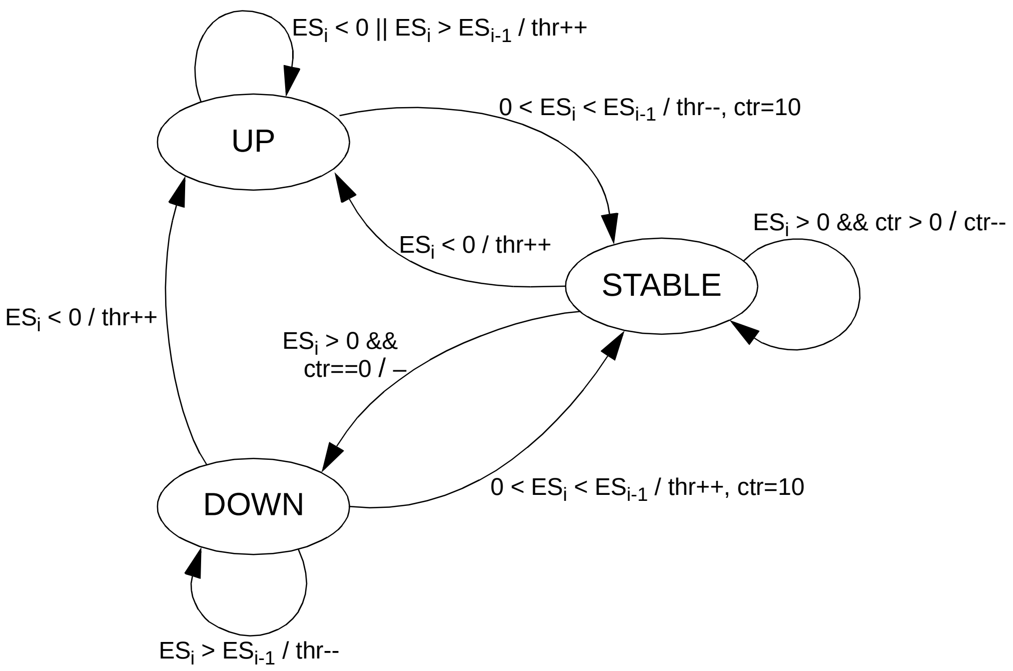
$ES_i < 0 \,||\, ES_i > ES_{i-1}$ / thr++
UP
$0 < ES_i < ES_{i-1}$ / thr--, ctr=10
$ES_i > 0$ && ctr > 0 / ctr--
$ES_i < 0$ / thr++
STABLE
$ES_i < 0$ / thr++
$ES_i > 0$ && ctr==0 / –
DOWN
$0 < ES_i < ES_{i-1}$ / thr++, ctr=10
$ES_i > ES_{i-1}$ / thr--

Figure 1. Energy-aware AMC control algorithm. ES_i is the estimated energy savings for interval i, thr is the decay threshold, and ctr is a counter used to stabilize the threshold once a minimum is found. In the UP or DOWN states, the threshold is adjusted in the appropriate direction as long as the energy savings increases ($ES_i > ES_{i-1}$). The UP state is entered immediately whenever a net energy loss is detected ($ES_i < 0$). Each threshold adjustment (thr++ or thr--) corresponds to a doubling or halving of the actual decay threshold value.

climbing strategy to converge iteratively on the threshold value which provides the minimum net energy consumption. Once we find a minimum-energy point, the threshold is held stable for ten intervals before restarting the search. If at any time the estimate shows a net energy loss, the algorithm immediately begins increasing the threshold until a new minimum is found. A detailed state machine for our control algorithm is shown in Figure 1.

Figure 2 compares the performance of this energy-aware AMC to that of the original AMC scheme for the SPEC CPU2000 benchmarks on a 1 MB, 8-way associative secondary cache. To maximize the range of available decay thresholds, we provide a generous 10-bit idle counter per block. These counters are incremented every 2048 cycles, as in [19], and cleared on an access to the block. (Additional details regarding our simulation methodology are provided in Section 5.) The energy-aware AMC scheme is significantly more effective than the original AMC at reducing net energy consumption. Perhaps more importantly, the energy-aware AMC algorithm never causes noticeable increases in energy consumption, unlike its energy-oblivious predecessor. Although the performance of the original AMC scheme can be varied by adjusting the target miss ratio, it is apparent from Figure 2 that the ratio of 1.5 used there is too large for some benchmarks (e.g., ammp, galgel, twolf), while reducing it will only reduce AMC's effectiveness for other benchmarks (e.g., crafty, perl). This result is unsurprising, as there is no direct correlation between this miss ratio and the net energy savings. Based on these results, we believe that any secondary-cache resizing scheme which makes resizing decisions without direct consideration of their energy impact is unlikely to insure a reduction in net energy consumption.

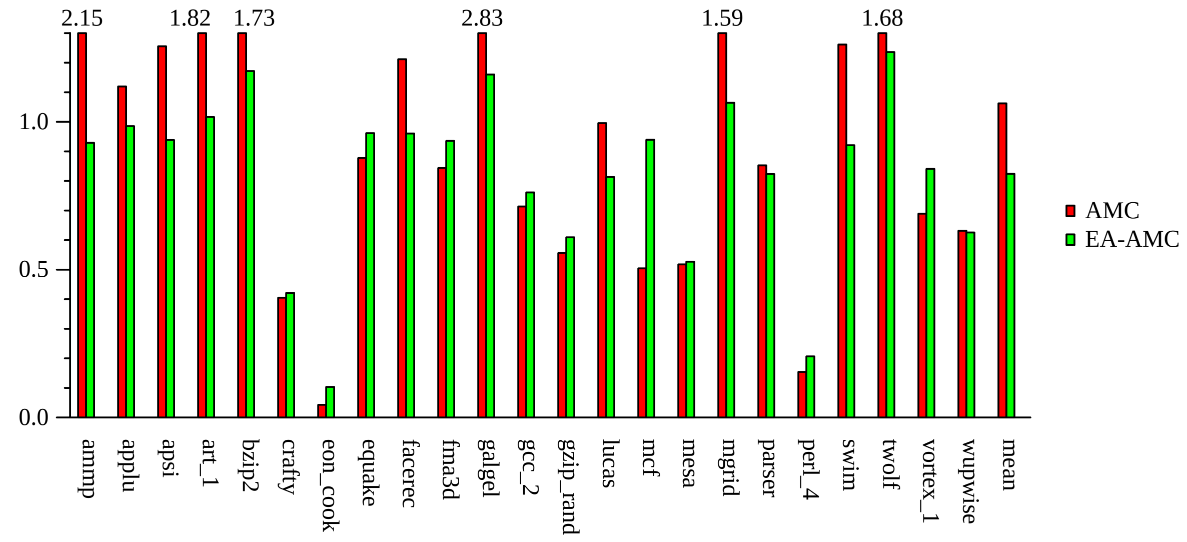

Figure 2. Normalized energy-delay product of original AMC and energy-aware AMC (EA-AMC).

While energy-aware AMC performs well, the iterative threshold adjustment process converges more slowly than necessary to the optimum setting. AMC's block-level control also incurs a noticeable hardware overhead in a large secondary cache: for our 1MB cache with 64-byte blocks, the 10-bit idle counters we used result in 160K bits of state; even 2-bit counters require 32K bits total. Although these overheads are but a small fraction of the total cache area, and thus quite tolerable, we present a new bank-level resizing scheme in the next section which provides similar overall performance with negligible hardware overhead.

4. Bank-Level Resizing (BLR)

In this section, we present *bank-level resizing (BLR)*, a control framework which combines resizing based on associative banks (ways) [2, 18] with per-way hit counters [15]. BLR trades the fine-grain block-level control of cache decay and AMC for two advantages. First, because BLR uses the same dimension—associativity—both to track misses and to resize, a control algorithm can calculate the value of any miss-based metric (such as net energy reduction) simultaneously at all possible resize points, not just at the current setting. Thus the optimal resize point can be selected directly after a single measurement interval, without iteration. Second, BLR adds only a small amount of hardware per associative way, with no per-block overhead. Thus BLR inherently has a much lower hardware overhead than block-level schemes, particularly for the large secondary caches on which we focus in this paper.

The key limitation of BLR is its coarse resizing granularity. As with other way-based resizing schemes [2, 18], BLR cannot tune the effective cache size to a non-integral multiple of banks. Also, as additional banks are disabled to reduce power, the effective associativity of the cache decreases, increasing the likelihood of conflict misses. BLR is thus inappropriate for low-associativity caches. However, most current on-chip secondary caches are at least four-way associative to minimize the performance penalties of off-chip accesses, and we expect future trends toward higher associativities. For example, the AMD Athlon [1] has a 16-way associative on-chip L2 cache.

BLR uses a counter for each logical way of the cache (MRU to LRU) to measure the number of accesses to that way (i.e., that LRU stack position). Because the logical replacement priority within a cache set is independent of the data's physical location, the selection of the appropriate counter to increment must combine the physical location of the tag match with the LRU state of the accessed set.[1] For enabled ways, the counter indicates the number of additional sleep misses that would have been incurred had the way been disabled. For disabled ways, the counter indicates the number of sleep misses that could have been saved had the way been enabled. In an n-way associative cache, the number of hits that would have occurred in a cache of any associativity $m \le n$ can be obtained by summing the m access counters closest to the MRU position.

We make two minor modifications to the LRU replacement scheme so that the way counters accurately reflect the number of hits that would have occurred in each way, even when some of the ways are disabled. First, on a miss, we replace the LRU enabled block, but evict the tag for the LRU *dis*abled block, and move the tag from the replaced LRU enabled block to become the MRU disabled tag.[2] Thus a subsequent access to the replaced block will be counted as a hit in the MRU disabled way. Similarly, a subsequent access to the former LRU disabled block will not be counted as a hit, as it would have been evicted even in a fully enabled cache. Second, when an access matches a disabled tag, we clear the tag (in addition to incrementing the associated way counter). This operation prevents multiple accesses to the same disabled block from counting as multiple independent misses that could have been avoided.

The maximum value of each counter is determined by the frequency with which the cache is accessed and the maximum interval at which the control algorithm samples (and resets) their values. Conservatively assuming a worst case of one access per way per cycle during a million-cycle interval leads to 20-bit counters; even with this level of overdesign, an 8-way associative cache has only 160 bits of counter state.

Although BLR's per-way access counts are based on the accessed block's logical replacement priority, for practical reasons BLR's power control must follow the physical cache structure. We use the term *bank* to indicate the physical memory structure associated with an associative cache way. There is no direct tie between the logical ways and physical banks of a cache; e.g., the MRU blocks in each set are effectively randomly distributed across the physical banks. When the BLR control algorithm decides to enable or disable an additional way, a random physical bank is chosen to be turned on or off. We experimented with copying data between physical banks when shrinking to ensure that only the logical LRU blocks were lost; however, this copying did not provide enough of a performance increase to warrant the extra dynamic energy expended or the complexity of the implementation.

In our current simulated implementation, all dirty blocks in a bank to be powered down are written back sequentially, stalling the processor. A more aggressive implementation would queue the writebacks and issue them in the background while the processor continues to execute. This scheme would reduce the impact of bank deactivations on performance. The impact on energy is less clear: by delaying writebacks, the bank may take longer to reach the fully powered-down state, but the energy savings due to the reduced runtime will counteract this effect.

Although we assume a true LRU replacement algorithm in this paper, we believe that BLR can be extended to deal with pseudo-LRU schemes, albeit with some loss of accuracy. The details will depend on the specific pseudo-LRU algorithm used; we leave this effort for future work.

1. Similar per-way access counters have been proposed by Suh et al. [15] for cache partitioning in multithreaded processors.
2. This operation can be performed without breaking the physical association between tag and data banks by physically copying the LRU enabled tag over the LRU disabled tag and updating the replacement priority appropriately.

```
T = period in cycles;
A = associativity of cache;
R = off-chip access to leakage energy ratio ($R_{OL}$);
threshold = T / (A * R);/* recalculated only when T or R is modified */
i = 0;
M = 0;

for (bank = first inactive bank; bank < number of banks; bank++){
   i++;
   M += hits[bank];
   if (M > (i * threshold) {
      turn on i banks, from first inactive bank to bank
      i = 0;
      M = 0;
   }
}

i = 0;
M = 0;
if (no banks turned on) {
   for (bank = last active bank; bank > 0; bank--) {
      i++;
      M += hits[bank];
      if (M < 0.8 * (i * threshold) {
         turn off i banks, from last active bank to bank
         i = 0;
         M = 0;
      }
   }
}
```

Figure 3. EnABLR algorithm pseudocode description.

4.1. Energy-Aware Bank-Level Resizing (EnABLR)

Modifying BLR to be energy aware is as simple as using Equation 3 for the resizing metric. For example, disabling i ways in a cache of associativity A reduces F_A by i/A. This change leads to a net energy reduction if the number of additional misses incurred by disabling those ways (call this M_i) over an interval of T cycles results in a net decrease in the estimate of Equation 3. That is, these i ways should be disabled only if

$$M_i < \frac{iT}{AR_{OL}} \qquad (4)$$

Figure 3 describes the EnABLR algorithm in pseudocode. As in Section 4, resizing decisions are normally made at regular intervals of T cycles. First, if any ways are disabled, we check whether they should be re-enabled. Specifically, we examine each disabled way in MRU to LRU order. For the ith disabled way, we compute M_i, the *sum* of the i way hit counters from the first disabled way up to the way in question. We apply Equation 4 to determine whether a net energy savings resulted from having them disabled. As soon as the algorithm detects a bank or set of banks which did not provide a net energy reduction, those banks are enabled, and the algorithm continues examining the remaining disabled banks (if any).

If no banks were enabled in the first phase of the algorithm, the second phase performs the complementary operation, examining the enabled banks from LRU to MRU to determine whether a net energy reduction would have resulted had they been disabled. To avoid thrashing, the disable threshold is set at 80% of the threshold for enabling a bank.

To enable more rapid adaptivity for a given resize interval *T*, EnABLR continuously compares the enable threshold (`threshold` in the pseudocode) with the hit counter for the first inactive way. If this counter exceeds the threshold, the bank is activated immediately.

One potential drawback of our design is that tag entries for disabled ways cannot be powered off, as the information they contain is vital to distinguishing the additional accesses due to resizing from the accesses that would be present with a full cache. If we only maintain tags for one additional way beyond the enabled portion of the cache, then we might still be able to adapt the cache size while disabling *n*-1 tag banks along with *n* data banks. Unfortunately, this approach caused problems in a few benchmarks. In *apsi*, the situation regularly arose where the number of misses saved by enabling one additional bank was too small to justify enabling that bank, but enabling *two* additional banks was amply justified due to the number of hits in the *second* disabled way. Leaving all tags enabled also allows us to enable multiple banks at a time and to turn on banks in the middle of a period, which helps to adapt more quickly to changing behavior and recover more rapidly from overly aggressive disabling.

5. Analysis

We simulated the EnABLR scheme using a modified version of SimpleScalar [6]. We configured the simulator to model an Alpha processor similar to the 21264, with out-of-order execution in a 64-instruction window, and up to 16 outstanding memory accesses. The memory hierarchy included split 64KB 2-way associative L1 caches, a 10-cycle 1MB 8-way set-associative L2 cache, and a 100-cycle latency to main memory. All caches used 64-byte lines and LRU replacement. We examine the full SPEC CPU2000 suite, to capture a wide range of benchmark behaviors.[1] Each benchmark was run for 200 million instructions from a checkpoint 20 billion instructions into the program's execution. We ran a subset of our experiments for 500 million instructions and saw no significant variation in the results.

We compare EnABLR with both the energy-aware AMC scheme, discussed in Section 3.3, and the cache decay scheme. As noted by Kaxiras et al. [11], we set the decay interval to the ratio of cache line leakage to an off-chip access energy to bound the energy loss. EnABLR and AMC use a resize interval of 1,000,000 cycles. The AMC decay clock ticks every 2048 cycles.

Figure 4 plots, for each benchmark, the energy dissipated, the execution time, and their product (normalized to the full-cache run) for an R_{OL} of 100. Values closer to zero are better, since they represent the greatest power savings. EnABLR achieves similar performance to both AMC and cache decay. The benchmarks where they greatly outperform EnABLR experience very few secondary cache misses, and require a very small cache size (less than one bank). In these cases, the finer block-level granularity provides a benefit. Both schemes are also able to turn blocks off in the middle of a adaptive interval, which EnABLR does not do. However, EnABLR never increases net system energy by more than 1%, and does not increase the normalized energy-delay product by more than 1%. The same cannot be said of either AMC or cache decay.

1. We were unable to simulate sixtrack and vpr due to errors in the simulation environment.

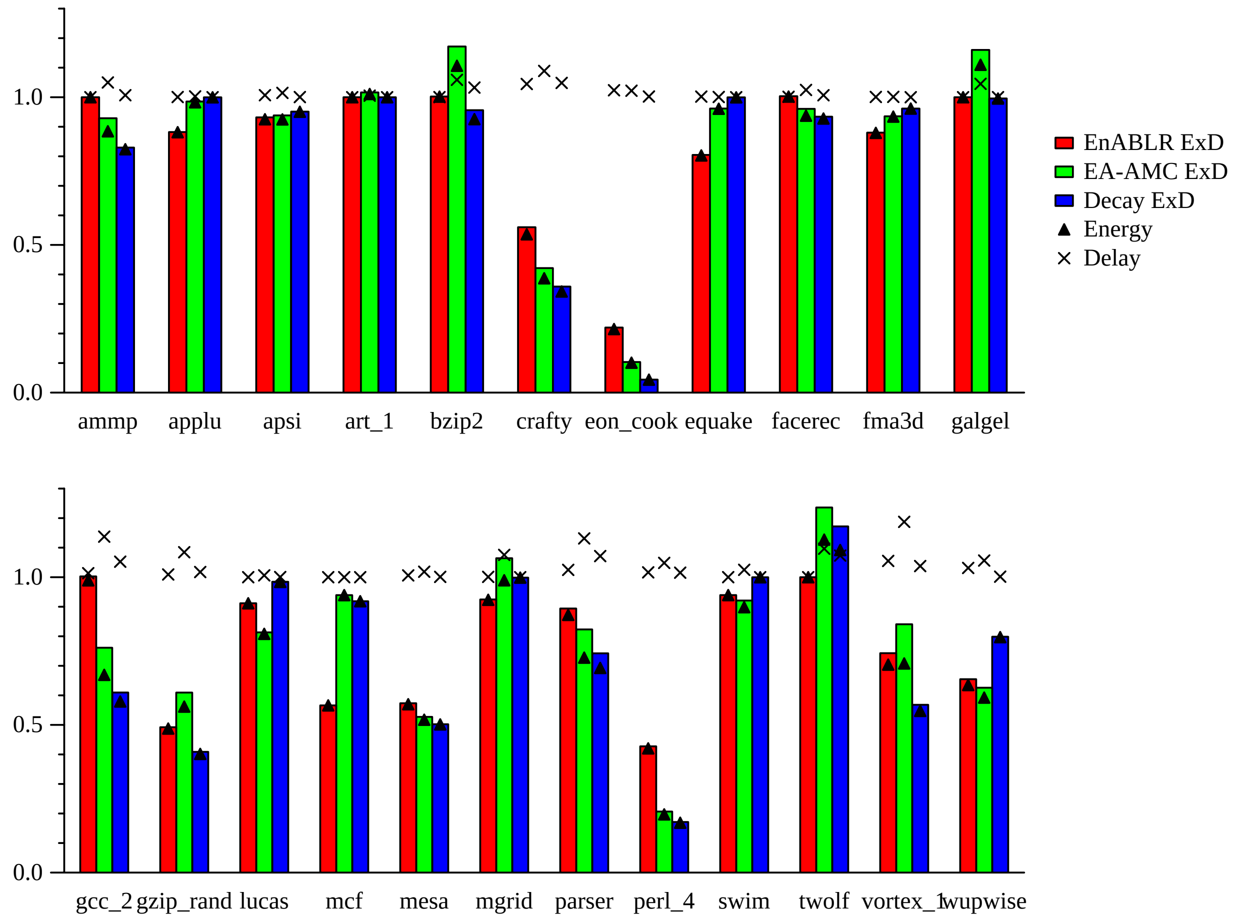

Figure 4. Relative energy, delay and energy-delay product for EnABLR, energy-aware AMC, and cache decay.

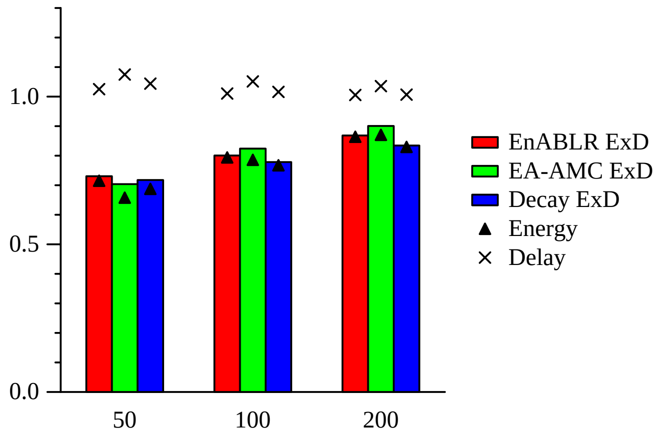

Figure 5. Mean energy, delay, and energy-delay product for R_{OL} = 50, 100, and 200

Breaking down the energy overheads of resizing, we found that extra misses, writebacks and execution time accounted for 78%, 19% and 3% of the overhead, respectively. These measurements indicate that tracking only additional misses, as in the EnABLR scheme, is indeed a reasonable approximation to estimate overall energy savings.

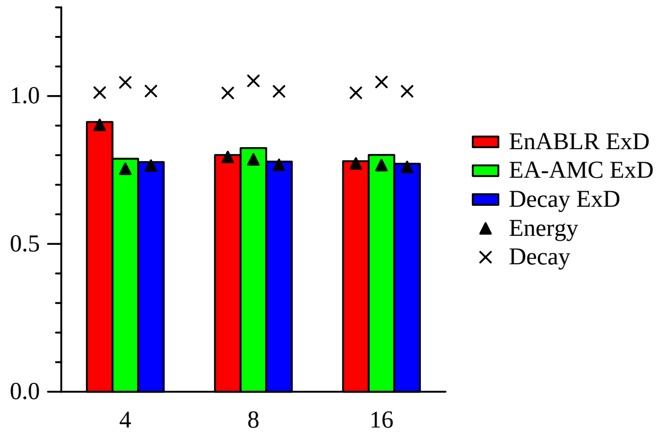

Figure 6. Mean energy, delay, and energy-delay product for associativities 4, 8, and 16 at R_{OL} = 100

To account for possible technology and operating temperature changes, we also ran for R_{OL} values of 50 and 200. Figure 5 presents the mean results over all the benchmarks for each of the three schemes. As can be clearly seen, EnABLR stays competitive at all ratios, and even outperforms AMC at the higher ratios.

Unfortunately, EnABLR's granularity changes as the associativity changes for a fixed size cache. Figure 6 shows the mean performance of the three schemes across all benchmarks for cache associativities of 4, 8, and 16. As expected, the block-level schemes suffer very little performance hit from lower associativities. EnABLR's performance seriously degrades with a 4-way associative cache. This is mainly due to the fact that a full quarter of the cache must be turned off at a time. However, EnABLR is competitive for 8-way and higher associativites, which is not atypical for secondary caches. Therefore EnABLR is a viable scheme for secondary cache leakage management.

6. Conclusions

Leakage energy in large on-chip caches is a significant issue for future microprocessors. This energy dissipation can be reduced by powering off unneeded portions of the cache. However, this leakage energy reduction must be carefully balanced with the induced energy costs of additional off-chip accesses and increased runtime. This paper highlights the importance of managing this trade-off explicitly; otherwise, cache resizing schemes may not only fail to reach a minimum net energy consumption, but may inadvertently increase overall system energy. We propose a practical formula for estimating the net energy savings, and apply it in the context of the previously proposed AMC resizing framework.

We also introduce a new bank-level resizing framework, which provides more complete information regarding the impact of cache resizing decisions at a much lower hardware overhead than block-based resizing schemes such as cache decay and AMC. Although the coarser resizing granularity limits the energy reduction in some extreme cases, an eight-way associative cache is adequate to provide comparable energy-delay performance.

The application of our energy-estimation formula in the BLR framework, called Energy Aware Bank-Level Resizing (EnABLR), provides energy reductions competitive with energy-aware block-level schemes with lower hardware cost. Across the SPEC CPU2000 benchmarks on a 1MB 8-way associative secondary cache, EnABLR reduces the overall energy used by the L2 cache by up to 80%. Average energy reductions range from 14% to 29%, depending on off-chip access energy costs.

Acknowledgments

This material was supported by the National Science Foundation under Grant CCR-9734026, by grants from Intel and Compaq, and by a Sloan Research Fellowship.

References

[1] Advanced Micro Devices, Inc. AMD Athlon processor and AMD Duron processor with full-speed on-die L2 cache. White Paper, June 2000. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/cache_wp.pdf.

[2] D. Albonesi. Selective cache ways: On-demand cache resource allocation. In *32nd Annual International Symposium on Microarchitecture*, pages 248–259, November 1999.

[3] R. Balasubramonian, D. Albonesi, A Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *33rd Annual International Symposium on Microarchitecture*, pages 245–257, December 2000.

[4] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic cache management techniques to reduce energy in a high-performance processor. In *Proceedings of the 1999 International Symposium on Low-Power Electronics and Design*, pages 64–69, August 1999.

[5] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.

[6] Doug Burger, Todd M. Austin, and Steve Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Technical Report 1308, Computer Sciences Department, University of Wisconsin–Madison, July 1996.

[7] J. Butts and G. Sohi. A static power model for architects. In *33rd Annual International Symposium on Microarchitecture*, pages 191–201, December 2000.

[8] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *To Appear in Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[9] F. Hamzaoglu, Y. Te, A. Keshavarzi, K. Zhang, S. Narendra, S. Borkar, M. Stan, and V. De. Dual-V_t sram cells with full-swing single-ended bit line sensing for high-performance on-chip cache in 0.13 um technology generation. In *Proceedings of the 2000 International Symposium on Low-Power Electronics and Design*, pages 15–19, August 2000.

[10] H. Hanson, V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D.C.Burger. Static energy reduction techniques for microprocessor caches. In *Proceedings of the 2001 International Conference on Computer Design*, pages 276–283, September 2001.

[11] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, July 2001.

[12] K. Nii, H. Makino, Y. Tujihashi, C.Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. H Hamano. A low power sram using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 International Symposium on Low-Power Electronics and Design*, pages 293–298, 1998.

[13] M. Powell, S. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 International Symposium on Low-Power Electronics and Design*, pages 90–95, August 2000.

[14] Srikanth T. Srinivasan and Alvin R. Lebeck. Load latency tolerance in dynamically scheduled processors. In *30th Annual International Symposium on Microarchitecture*, pages 148–159, November 1998.

[15] G.E. Suh, S. Devadas, and L. Rudolph. A new memory scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA)*, February 2002.

[16] Dennis Sylvester. Personal communication, October 2001.

[17] S. 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. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 147–157, January 2001.

[18] S. Yang, M. Powell, B. Falsafi, K. Roy, and T. Vijaykumar. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 147–157, February 2002.

[19] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, September 2001.