Data Cache Techniques to Save Power and Deliver High Performance in Embedded Systems

Major Bhadauria[1], Sally A. McKee[1], Karan Singh[1], Gary S. Tyson[2]

[1] Computer Systems Lab
School of Electrical and Computer Engineering
Cornell University
{major | sam | karan}@csl.cornell.edu
[2] Department of Computer Science
Florida State University
tyson@cs.fsu.edu

Abstract. Minimizing power consumption continues to grow as a critical design issue for many platforms, from embedded systems to CMPs to ultrascale parallel systems. As growing cache sizes consume larger portions of the die, reducing their power consumption becomes increasingly important. Voltage scaling reduces leakage power for cache lines unlikely to be referenced soon. Partitioning reduces dynamic power via smaller, specialized structures. We introduce a *reuse distance* (RD) drowsy caching mechanism that exploits temporal locality, delivers equivalent or better energy savings than the best policies from the literature, suffers little performance overhead, is simple to implement, and scales with cache size and hierarchy depth [3].

1 Introduction

Minimizing power consumption continues to grow as a critical design issue for many platforms, from embedded systems to CMPs to ultrascale parallel systems. Cache sizes have grown steadily in an attempt to mask the widening gap between main memory latency and core clock frequency and to avoid the large energy costs of off-chip memory accesses [11]. Caches thus consume increasing portions of die area (approximately half for current general-purpose chips) and account for larger percentages of total system power. Most of the increase stems from exponential growth in leakage due to shrinking feature sizes and their decreasing transistor voltage thresholds. The bottom line is large caches cause significant current leakage. We investigate power issues for embedded and higher performance systems and present solutions tailored for each domain.

Previous research targets both static (leakage) power and dynamic switching power. Leakage power can be reduced by making caches smaller or shutting off portions. Another option is to reduce the operating voltage for those portions, which reduces leakage current but increases access latency. Such drowsy

[3] This paper extends an article from the HiPEAC 2007 conference.

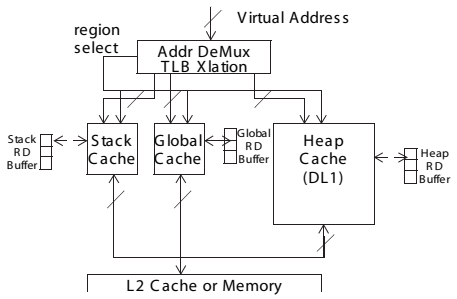


Fig. 1. Organization of Region-Based L1 Caches

caches [5] usually reduce voltage on all cache lines periodically, "waking up" (bringing back to normal voltage) lines as they are accessed. Since most lines remain idle most of the time, drowsy caching techniques can help reduce leakage power. Some mechanism must implement a policy to control when to turn lines off and on. Two main policies have been studied previously: the *simple policy* indiscriminately puts all lines to sleep after a specified number of clock cycles (and is thus clock-frequency dependent with respect to some performance aspects), and the *noaccess policy* only turns off lines that have not been accessed within a predefined number of cycles (also clock-frequency dependent). Simple performs almost identically to the more sophisticated noaccess, and thus is most often used in drowsy cache organizations [5]. Petit et. al [19] propose a *Reuse Most Recently used On* (RMRO) drowsy cache that behaves much like a noaccess policy adapted for associativity, using update intervals to calculate how often a set is accessed. The MRU way remains awake, but other ways of infrequently used sets are made drowsy. RMRO requires more hardware per set and uses dynamic power every cycle. These policies all count elapsed clock cycles to determine when cache lines should be put to sleep, making them inherently architecture-specific, and preventing prediction of leakage savings before individual workload evaluation, which we find to be a drawback.

Switching power can also be minimized by reducing the size of the cache. Unlike leakage power, which is determined by the total area of all powered portions of the cache, switching power is only consumed by the portion of the cache being accessed. Partitioning can thus reduce switching power. For instance, a single cache can be partitioned into multiple sub-cache structures that are accessed independently. When a reference is made, only one of the cache partitions is accessed; the selection of which cache to access depends on the partitioning strategy. Region caches [18, 16, 17, 6] partition the data cache into separate stack, global and heap caches that service accesses to the corresponding memory regions. The heap cache serves as a general L1 Dcache for data not assigned to other caches. A simple address decoder (using only a few address bits) identifies the region and routes the reference to the appropriate cache.

Our research builds on both drowsy caching and region caching. We develop a new region cache model that maintains original hit rates while reducing leak-

age via smaller multi-access column associative (CA) [1] and MRU [10] caches. These structures exhibit excellent implementation cost, performance, and energy tradeoffs for the target applications. We employ a drowsy policy that is simple to implement, scales better than previous policies, and allows finer control over cache power management. This *Reuse Distance* (RD) policy tracks lines accessed, keeping a limited (potentially configurable) number of lines awake. Fig. 1 illustrates our organization (RD buffers are much smaller than depicted). The RD buffer records IDs of awake cache lines. When the buffer is full and a new cache line is accessed, the LRU line is made drowsy and the new line's ID is entered. A buffer of N entries needs N counters of $log_2 N$ bits, since they're incremented every memory access (not per cycle, as in most previous drowsy policies).

Power consumption is a first order design criteria for most embedded systems. Therefore, we adapt techniques originally proposed to improve cache *performance* to develop robust cache organizations to conserve energy in high performance embedded systems. On the MiBench suite, our multiple-access associative caches with RD drowsiness deliver IPCs of more complicated drowsy designs while using 16% less power. Compared to non-drowsy designs, we reduce power by 65% on average, while remaining within about 1% of the original mean IPC.

We then examine using RD for higher performance systems with more traditional cache organizations. For these systems, we solely examine drowsy behavior (previous studies have examined region caches in similar settings). We compare RD and simple drowsy policies for 25 SPEC CPU 2000 benchmarks and a range of hierarchies. For modest clock rates of 0.5-1.5 GHz we observe IPC decreases of 2.4% on average over non-drowsy caches, in exchange for power savings of 93% for 32KB L1 caches and 95% for 512KB L2 caches, respectively. The latter represents a percentage of a much larger number: a significant energy savings. As L2 grows relative to L1, we maintain both performance and power savings by keeping fewer L2 lines awake. Finally, we enable larger overall cache sizes and/or higher clock rates, delivering better performance while adhering to a strict power budget. Ultimately, we propose data cache organizations that better match reference behavior and provide finer control for dynamic power management.

2 Related Work

Inoue et al. [11] survey techniques for reducing memory access energy while maintaining high performance. We discuss those techniques most related to our investigations: partitioned, drowsy, and multi-access cache organizations.

Breaking monolithic memories into separate components enables optimizing those components to achieve better performance or conserve energy. Early "horizontal" partitioning broke L1 caches into separate instruction (Icache) and data (Dcache) designs, and "vertical" partitioning added multiple levels to the hierarchy. Modern vertical partitionings may create an L0 memory level between the processor and L1, as in line buffers [12, 8] and filter caches [15]. These small structures reduce average access energy for hits at the expense of increased access latency for misses (which increases average L1 latency). Horizontal partitioning

reduces dynamic power by directing accesses at the same hierarchy level to different structures or substructures, as in cache subbanking [8], allowing smaller structures. Specialized loop caches and scratchpad memories can improve performance for scientific or embedded applications, for instance. Since different data references exhibit different locality characteristics, breaking the L1 data cache into separate, smaller horizontal structures for stack, global, and heap accesses [16, 6] better exploits locality behavior of the different data *regions*. This can improve hit rates and cut dynamic and static energy consumption.

Turning off dead portions of cache after writing dirty lines back to the next level of memory helps control leakage energy. Such *decay caches* [20, 13] often trade performance for reduced power consumption. By predicting dead lines, Kaxiras et al. [13] reduce L1 leakage energy by up to a factor of five with little performance impact. A two-bit counter per line tracks the current working set, and at each adaptive decay interval they set line states based on counter values.

Even when cache lines are still live, they may be read infrequently. Placing idle cache lines in a dormant state-preserving (*drowsy*) condition reduces static power consumption by dynamically decreasing supply voltage of the wordlines. Drowsy lines must be brought back to normal voltage before being loaded in the sense-amplifiers, thus access latency to these lines is increased. Repeatedly changing state causes high performance and power overheads. Drowsy wordlines consist of a drowsy bit, voltage control mechanism, and wordline gating circuit [14]. The drowsy bit controls switching between high and low voltages, and the wordline gate protects data from being accessed in drowsy mode. Flautner et al. [5] investigate *simple* and *noaccess* strategies (described in Sect. 1) to keep most lines drowsy and avoid frequent state changes. Geiger et al. make their heap cache drowsy [7] and explore a small, non-drowsy *hot heap cache* to hold lines with high temporal locality [6]. Since the stack and global caches service most references, an aggressive drowsy policy in the heap cache has negligible performance effects. The RMRO policy of Petit et al. [19] (described in Sect. 1) makes associative cache ways drowsy depending on their usage (this attempts to reduce leakage similarly to Albonesi's [2] selective cache ways for dynamic power reduction). These policies cannot place a hard limit on total cache leakage.

A two-way associative cache delivers similar performance to a direct mapped cache twice the size, but even on accesses that hit, the associative cache wastes power on the way that misses: two banks of sense-amplifiers are always charged simultaneously. *Multiple-access* or *Hybrid Access* caches address the area and power issues of larger direct mapped or conventional two-way associative caches by providing the benefits of associativity—allowing data to reside at more places in the cache, but requiring subsequent lookups at *rehashed* address locations on misses—to trade a slight increase in complexity and in average access time for lower energy costs. Smaller structures save leakage power, and charging shorter bitlines or fewer sets saves dynamic power. Examples include the hash-rehash (HR), column associative (CA) [1], MRU [10], skew associative [21], and predictive sequential associative [4] caches. HR caches swap lines on rehash hits to move most recently accessed lines to original lookup locations. CA caches avoid

Process	70nm
Operating Voltage	1.0V
Operating Temperature	65^o C
Frequency	1.7 GHz
Fetch Rate	1 per cycle
Decode Rate	1 per cycle
Issue Rate	1 per cycle
Commit Rate	1 per cycle
Functional Units	2 Integer, 2 FP
Load/Store Queue	4 entries
Branch Prediction	not taken
Base Memory Hierarchy Parameters	32B line size
L1 (Data) Size	32KB, direct mapped
L1 (Data) Latency	2 cycles
L1 (Instruction) Size	16KB 32-way set associative
L1 (Instruction) Latency	1 cycle (pipelined)
L2(Unified) Size	512KB 4-way set associative
L2 Latency	12 cycles
Main Memory	88 cycles

Table 1. Baseline Processor Configuration

L1 (Data) Size	16KB, CA or 2-Way Set Associative
L1 (Data) Latency	1 cycle
L1 (Stack)Size	4KB, direct-mapped
L1 (Stack) Latency	1 cycle
L1 (Global) Size	4KB, direct-mapped
L1 (Global) Latency	1 cycle
Drowsy Access	1 cycle

Table 2. Region Drowsy Cache Configuration Parameters

thrashing between original and rehash locations by adding a bit per tag to indicate whether a given line's tag represents a rehashed address. Adding an MRU bit for way prediction reduces power consumption with minimal performance effects. Ease of implementation and good performance make CA and MRU way-predictive structures attractive choices for low power hierarchies; other multiple-access designs focus on performance over energy savings.

3 Experimental Setup

We use SimpleScalar with HotLeakage [24] for the ARM and Alpha ISAs. For the embedded platform we simulate the MiBench suite [9], which represents a range of commercial embedded applications. For the high performance sector, we evaluate SimPoints [22] for 25 SPEC CPU 2000 applications with reference inputs on a four-issue Alpha architecture.

3.1 Embedded Architecture

Our simulator models region caching [6], and we incorporate column associativity and MRU way prediction, along with our RD drowsy policy. The embedded processor is single-issue, in-order, and five-stage pipelined, (see Table 1). We calculate static power consumption via HotLeakage, using CACTI 3.2 [23] and Wattch [3] to calculate dynamic power. We calculate memory latency for single accesses via CACTI. For baseline comparison with Geiger et al.'s work [7] we use their direct-mapped cache configurations (see Table 2).

HotLeakage calculates static cache leakage as a function of process technology and operating temperature. Operating temperature is 65^o C, suitable for embedded systems for personal electronics. Dynamic consumption for typical logic

Technology	70 nm
Frequency	1.5 GHz
Temperature	80 C
Voltage	1 V
Issue/Decode/Commit Width	4
Instruction Fetch Queue Size	8
INT/FP ALU Units	4/2
Physical Registers	80
LSQ	40
Branch Mispredict Latency	2
Branch Type	Tournament
L1 Icache	32KB 4-Way Associative 1-cycle Access 32B Lines
L1 Dcache	32KB 4-Way Associative 1-cycle Access 32B Lines
L2 Cache	256KB/512KB/1MB/2MB 4-way Associative 4/10/27/32 cycles 32B Lines
Main Memory	97 cycles

Table 3. High Performance Architectural Parameters

circuits is $\frac{1}{2}CV^2f$, a function of frequency, capacitance and operating voltage. CACTI accurately calculates numbers of sense amplifiers in associative caches. We convert CACTI energy numbers to power values based on parameters in Table 2. CACTI dynamic power values combined with HotLeakage static values compute total power, scaling CACTI results for frequency to add accurate numbers from different tools. Drowsiness or IPC changes do not affect dynamic power (e.g., from scaling clock rates), since it is independent of runtime. We assume negligible dynamic power increases due to clock switching for lower IPCs.

Fig. 2 illustrates organization of a Reuse Distance drowsy mechanism. RD tracks lines being accessed, keeping a limited number of lines awake. The RD buffer stores IDs corresponding to awake cache lines. When the buffer is full and a new cache line is accessed, the LRU line is made drowsy and its ID is overwritten by the new line's. Counters track the LRU buffer entry. RD circuitry never dictates what lines to awaken, but only lines to make drowsy, which keeps it off the critical path (making RD timing irrelevant). Power consumption and silicon area are accounted for (but negligible). An RD buffer of N entries only needs N counters of log_2N bits that are updated every memory access (and not every cycle). Assuming a reuse distance of eight and 1024 cache lines (as for a 32KB 32-Byte line baseline cache), storing awake cacheline IDs and LRU counts requires 104 bits ($[log_2 1024 + log_2 8$ bits$] * 8$), of which only a single buffer's count (three bits) is reset every cache access. Power consumption for this extra circuitry is akin to the simple policy's single counter's dynamic power, since more bits are used but are accessed less frequently. An alternative implementation replaces counters with timestamps (incremented every cycle) to record when cache lines are accessed, and the lowest indicates the LRU line. To keep error of such an LRU scheme low, sufficient timestamp bits are needed (32 or 64 bits).

	Cache Accesses Check These Bits	Drowsy Misses Increment These Bits
0	124	3
1	11	4
2	325	7
3	804	0
4	806	2
5	125	6
6	803	5
7	805	1

Fig. 2. Organization of Drowsy LRU Structure for an RD of Eight

3.2 High Performance Alpha Architecture

Table 3 lists parameters of our Alpha 21264 model. For a valid baseline comparison with Flautner et al.'s work, we use their cache parameters, instead of the real 21264's. All simulations use separate, single-cycle access, 32KB direct-mapped instruction and 32KB four-way associative data L1 caches. The unified L2 is four-way set-associative and 256KB, 512KB, 1MB or 2MB in size (with appropriate memory latencies). We model drowsy L1 data and L2 caches. Switching from between sleep and wake-up modes incurs a one-cycle transition penalty. Tag lines stay awake, so only hits to drowsy cache lines suffer extra latency.

Leakage current is a function of process technology (due to the transistor voltage threshold) and is largely dependent on temperature (doubling approximately every 10 degrees). We model an operating temperature of 80^o C (typical for personal electronics), and a 70nm[4] process technology. These are sufficiently state-of-the-art to support future cores with large L2 caches. Table 4 reports power parameters extracted via Wattch for modest clock frequencies of 500MHz and 1.4GHz. RD5 indicates that we keep five lines awake (in the L1 Dcache); RD1 indicates that we keep a single line awake (here, in the L2 cache). Using RD5 (for L1) and RD1 (for L2) policies reduces leakage by 92% and 94%, respectively. This configuration lets us maintain constant power while increasing core frequency from 500MHz to 1.4GHz. We account for power consumed by all cache circuitry, including mechanisms to implement RD drowsy caching.

4 Evaluation

We apply simple and RD policies to L1 data caches, examining leakage and power performance for embedded and high performance architectures. We extend the embedded architectures with multiple-access caches (half size) for the heap region. For the high performance architecture, we apply drowsiness to L1 and L2 caches to maximize energy reductions with minimal performance degradation. We compare performance as L2 scales and demonstrate how the power envelope can be used for scaling frequency. We compare RD and simple drowsy policies for different architectures, modifying parameters to obtain best energy-delay tradeoffs for each architecture. We characterize reuse distance for all benchmarks, finding the appropriate RD to maximize temporal locality for each suite.

[4] This is the smallest feature size for which CACTI has an accurate spice model.

Leakage Power	
Non-Drowsy Caches	
L1 D-Leakage (32KB)	0.134
L2 D-Leakage (2MB)	8.827
Ireg	0.002
I-Cache	0.131
Core Processor Leakage Assumed	4
Total Leakage	9.960
Drowsy Caches	
L1 D-Leakage (32KB) (Drowsy RD5)	0.011
L2 D-Leakage (2MB) (Drowsy RD1)	0.530
Ireg	0.002
I-Cache	0.130
Core Processor Leakage Assumed	4
Total Leakage	1.663
Dynamic Power	
500MHz Core Clock Frequency	
Total Chip Dynamic Power	7.502
Total Chip Power	20.461
1.4GHz Core Clock Frequency	
Total Chip Dynamic Power	16.159
Total Chip Power	20.822

Table 4. Benchmark-Independent Power Parameters (Watts) for Frequency Scaling

4.1 Embedded Architecture

We compare our baseline region caches to organizations where the heap cache is replaced with a multiple-access cache (CA or MRU) of half the size. Access patterns for stack and global caches make their direct mapped organizations work well, thus they need no associativity. We study drowsy and non-drowsy region caches, comparing simple, noaccess, and RD policies. Keeping all lines drowsy incurs an extra cycle access penalty lowering IPC by up to 10% for some applications. If performance is not crucial, completely drowsy caches are attractive design points. The noaccess drowsy policy always uses slightly more power than the simple policy, but it yields higher average IPC by about 0.6%. Given the complexity of implementation, low payoff in terms of performance, and lack of energy savings, we use the simple policy in our comparisons.

Keeping only three to five lines awake at a time yields good results. Many applications reuse few lines with high temporal locality, while others have such low temporal locality that no drowsy policy permits line reuse: our policy works well in both cases. The RD drowsy policy is implemented with a buffer (per region cache) that maintains an N-entry LRU "cache" of the most recently accessed set IDs. The RD buffer LRU information is updated on each cache access, and when a line not recorded in the buffer is accessed and awakened, the LRU entry is evicted and put to sleep. Unlike the simple and noaccess policies, RD requires no counter updates or switching every clock cycle, and thus consumes no dynamic power between memory accesses. We approximate RD's dynamic power consumption as the number of memory accesses multiplied by the switching power of a number of registers equal to the buffer size. Static power overhead is negligible relative to cache sizes. RD is essentially a simplified implementation of the noaccess policy, but is based on the last unique N lines, not lines accessed during an arbitrary update interval. We experiment with buffers of three, five,

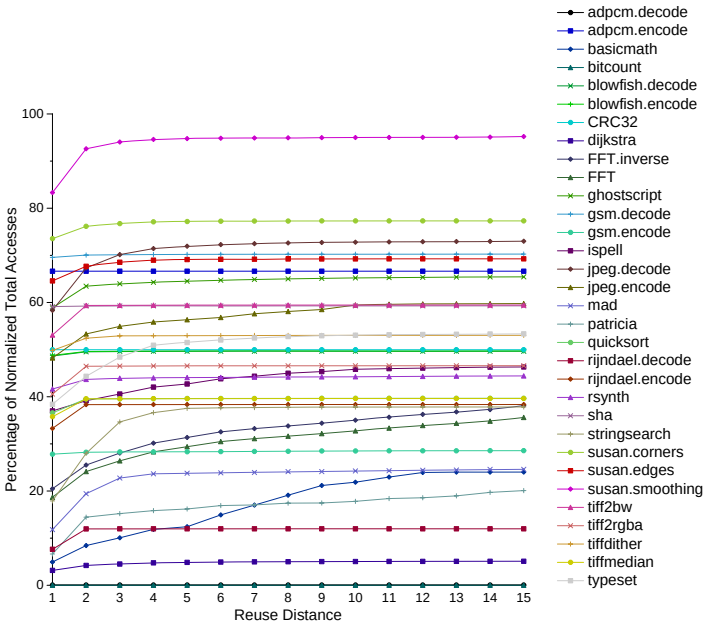


Fig. 3. Reuse Distances for Last 15 Cache Lines

10, and 20 entries, finding that buffers of only three entries reduce power significantly over those with five, while only negligibly decreasing IPC. Larger buffers suffer larger cache leakage penalties because more lines are kept awake, in return for reducing the number of drowsy accesses. Fig. 3 illustrates the percentage of total accesses that are accessed within the last N or fewer memory references. On average, the last three and four memory accesses capture 43.5% and 44% of the total accesses to the heap cache. The increase is not linear, as the last 15 unique memory line accesses only capture 45.5% of all heap accesses. Update window size for the other drowsy policies is 512 cycles for the heap and stack cache, and 256 cycles for the global cache[5]. We find RD's power and performance to be highly competitive with the simple policy for the caches we study. Note that our small update windows are aggressive in their leakage energy savings: windows of 4K cycles, as in Flautner et al. [5] and Petit et al. [19], suffer 20% more leakage energy. Unlike RD, the simple drowsy policy is dependent on update window size and CPU frequency. This means RD scales well with number and sizes of caches: hardware overhead is minimal and fixed.

To provide intuition into simple drowsy policy performance and motivation for our RD configuration, we track the number of awake lines during update intervals. Fig. 4 shows that for all benchmarks, on average, 66% of the intervals access fewer than four lines. These data are normalized to the total number

[5] These values were found to be optimal for simple drowsy policies and region caches [7].

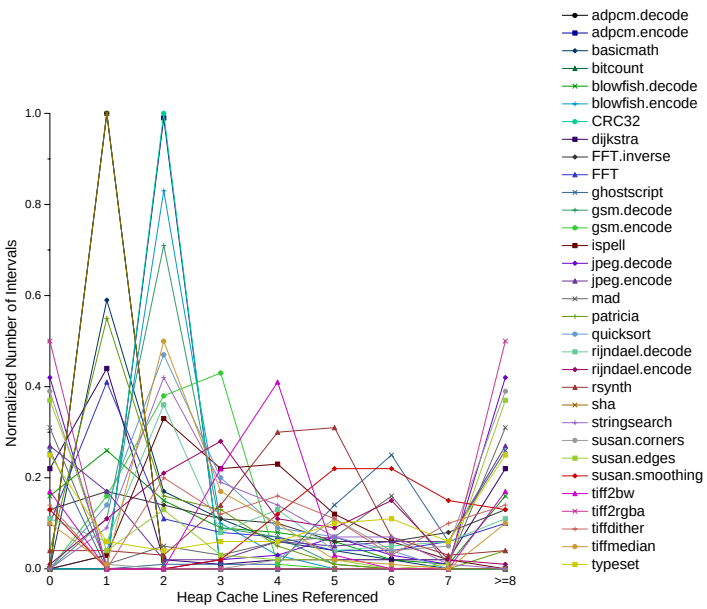
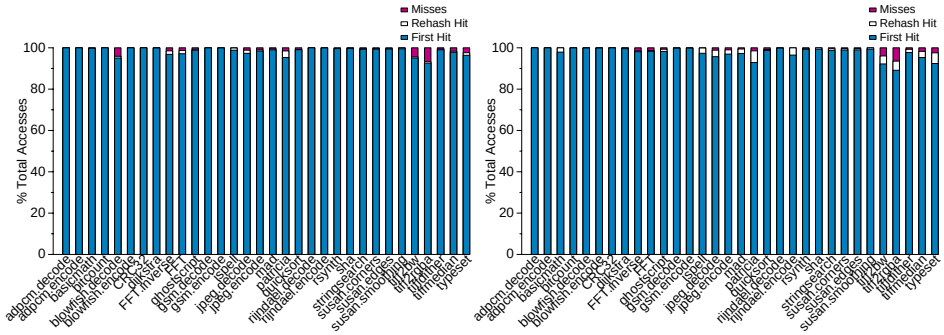


Fig. 4. Number of Heap Cache Lines Accessed During *simple* Intervals (512 Cycles)

of intervals per benchmark so that detail for shorter-running benchmarks is preserved. Benchmarks such as jpeg.decode use eight or more lines during 42% of their intervals, but larger active working sets mean increased leakage. RD's performance on jpeg.decode is within 2% of simple's, and RD saves 5% more leakage energy by limiting the number of lines awake at a time. These examples indicate that a configurable RD buffer size would allow software to trade off performance and energy savings: systems or applications would have fine-grain control in enforcing strict performance criteria or power budgets.

Although CA organizations potentially consume higher dynamic power on a single access compared to a direct mapped cache, this slight cost is offset by significant leakage savings since the CA cache is half the capacity. The CA organization consumes less dynamic power than a conventional two-way set associative cache that charges two wordlines simultaneously (the two-way consumes the same power on hits and misses). In contrast, a CA cache only charges a second line on a rehash access. The second lookup requires an extra cycle, but rehash accesses represent an extremely small percentage of total accesses. Fig. 5 shows percentages of accesses that hit on first lookup, hit on rehash lookup, or miss the cache: on average, 99.5% of all hits occur on the first access.

MRU associative caches use a one-bit predictor per set to choose which way to charge and access first. This performs well because most hits occur to the way last accessed. On a miss to both ways, the prediction bit is set to the way holding the LRU line to be evicted. On an incorrect guess or miss, MRU caches suffer an extra cycle of latency over a normal two-way associative cache. This is



(a) Column Associative Cache (b) MRU Cache

Fig. 5. Heap Accesses Broken Down by Category

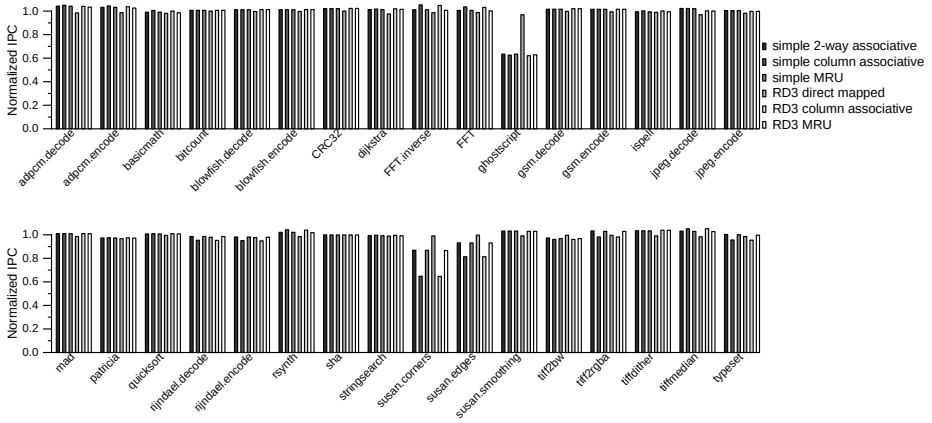


Fig. 6. IPCs Normalized to *simple* Drowsy Direct Mapped Region Caches

offset by significant power savings on correct predictions: since the MRU cache is physically partitioned into two sequential sets, it only charges half the bitline length (capacitance) of a same size direct mapped or CA cache. Fig. 5 shows percentages of hits in the predicted way, hits in the second way, and misses. On average, 98.5% of all hits are correctly predicted, resulting in a 50% reduction in dynamic power. The remaining accesses (the other 1.5% that hit plus the misses) consume the same dynamic power as a two-way associative cache.

4.2 Sustaining Performance

Fig. 6 graphs IPCs relative to direct mapped caches with the best update windows from Geiger et al. [7]. For most benchmarks, two-way set associative, CA, and MRU caches match or exceed the performance of their direct mapped counterparts, but do so at half the size. Hit rates are competitive, and MRU caches afford smaller access latencies from correct way predictions. Exceptions are ghostscript and the susan image processing codes, which have higher hit

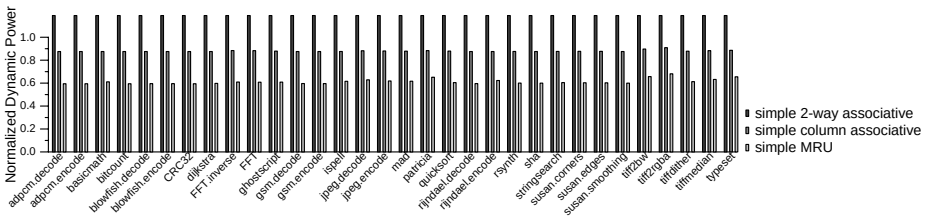


Fig. 7. Dynamic Power Consumption of Associative Heap Caches Normalized to a Direct Mapped Heap Cache

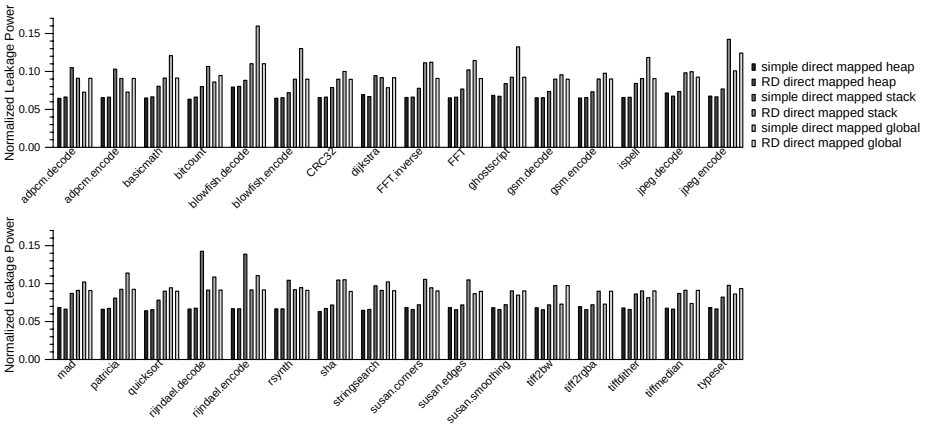


Fig. 8. Static Power Consumption of Different Drowsy Policies with Direct Mapped Region Caches Normalized to Non-Drowsy Direct Mapped Region Caches

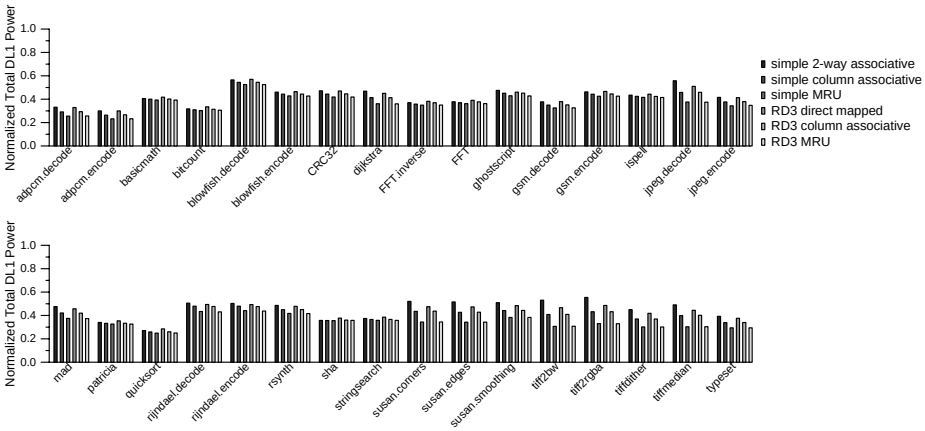


Fig. 9. Total Power of Drowsy Policies (for All Region Caches) and Heap Cache Organizations Normalized to Direct Mapped Region Caches

rates with associativity, but lower IPCs.[6] Overall, IPCs are within 1% of the best baseline case, as with the results of Flautner et al. [5]. These differences fall within the range of modeling error.

4.3 Reducing Dynamic Power

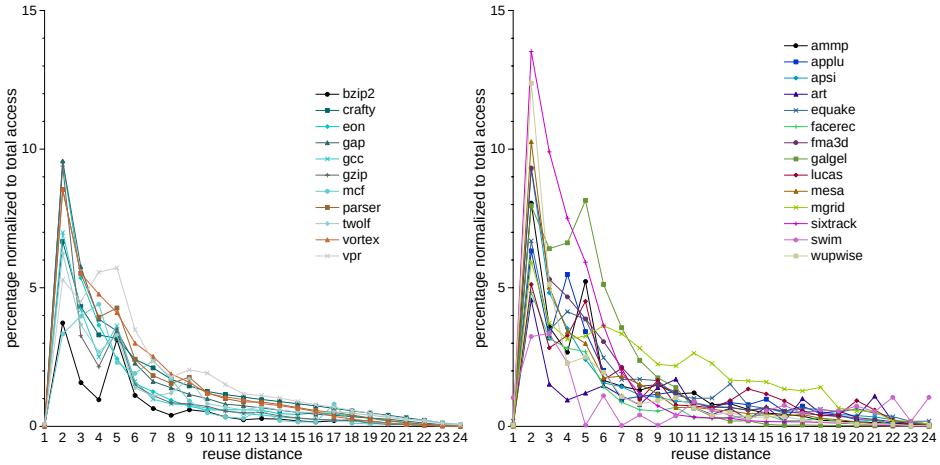
Fig. 7 shows heap cache dynamic power normalized to a direct mapped baseline (note that dynamic power is independent of drowsy policy). CACTI indicates that the two-way associative cache uses 19% more power, the CA cache uses 7.3% less power, and the MRU cache uses 50% less power on a single lookup. Fig. 7 illustrates this for CA and MRU organizations: rehash checks and way mispredictions increase power consumption for some scenarios.

4.4 Reducing Leakage Current

Access pattern has little effect on static leakage. However, static power consumption is higher for benchmarks for which associative organizations yield lower IPCs than the direct mapped baseline. Reducing sizes of associative caches reduces leakage on average by 64% over the baseline direct mapped caches. Although IPC degrades by 1% between non-drowsy and drowsy caches, the leakage reduction in drowsy organizations is substantial, as shown in Fig. 8. Performance for the RD policy is slightly worse, but differences are sufficiently small as to be statistically insignificant. The noaccess policy has highest IPCs, but suffers greatest leakage, dynamic power, and hardware overheads; we exclude it in from our graphs to make room for comparison of the more interesting design points. Simple and RD exhibit similar savings, but the RD mechanism is easier to implement and tune, without requiring window size calibration for each workload. Software-configurable update window sizes for simple drowsy policies and RD buffer sizes achieve better power-performance tradeoffs among different workloads. No single setting will always yield best results.

The effects of heap cache dynamic energy savings shown in Fig. 7 represent a small contribution to the total L1 Dcache power consumption shown in Fig. 9. Implementing drowsy policies across all memory regions plays a significant role in the power reductions we observe, with RD again having lowest power consumption of the different policies. MRU caches implementing RD drowsiness yield the lowest power consumption of the organizations and policies studied. This combination delivers a net power savings of 16% compared to the best baseline region organization implementing simple drowsiness, and 65% compared to a typical non-drowsy, partitioned L1 structure. These significant power savings come at a neglible performance reduction of less than 1.2%.

[6] We find these results to be anomalous: on a single-issue, in-order, architecturally similar Alpha simulator, the way-associative caches have higher IPCs for these benchmarks.



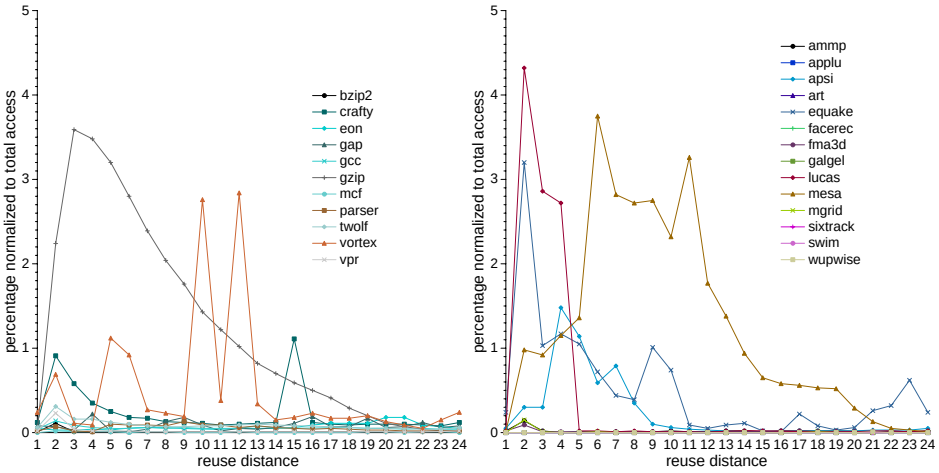
(a) Integer Applications (b) Floating Point Applications

Fig. 10. L1 Temporal Locality

4.5 High Performance Architecture

For high performance architectures, the simple policy uses an optimal 4000-cycle execution window [5] before all cache lines expire and are turned off. Fig. 10 and Fig. 11 show histograms of the percentages of total cache accesses having reuse distances from one to 24 line accesses. We use this to find good power-performance tradeoffs for our RD mechanism. Temporal locality drops significantly after the most recent five unique line references. Temporal locality for the L2 is an order of magnitude lower than the L1, most likely due to the larger size, which results in many more lines to which data may map (lines $\times$ associativity, i.e., 4096 $\times$ 4). After the last five cache lines are accessed, locality is very low for most benchmarks (vortex, vpr and mesa being the exceptions). This indicates that larger RD buffers will provide little benefit. Based on our histogram analysis, we partition the RD to keep 15 lines awake in L1 and one in L2 (i.e., the most recently accessed line). A drowsy L1 access increases latency significantly (by 50%) compared to a drowsy L2 access. We therefore spend the majority of our power budget masking performance degradation at the L1 level. We simulate an RD of size one, five and 15 and find 15 to give the best ratio of performance to active lines. Differences between RD policies are accentuated in Fig. 12, which shows numbers of drowsy accesses normalized to an RD of one, averaged for all the benchmarks. An RD of five decreases drowsy accesses by 35%, and an RD of 15 by 60%. The rate of return decreases after an RD of 15, so we use that as our basis for comparison with other drowsy policies.

Fig. 13 through Fig. 15 compare IPCs and leakage for RD versus simple and RMRO [19] for a 512KB L2. Both deliver significant leakage savings with almost indiscernible performance degradation (less than 3%). RD delivers substantial savings at the L1 level (55.2% more than simple and 69.6% more than RMRO),



(a) Integer Applications (b) Floating Point Applications

Fig. 11. L2 Temporal Locality

since the number of awake lines is capped at 15 at any time. RD achieves lower power consumption than simple with the L2 cache, but the improvement is not as high as in the L1, since the L2 is not accessed as often: most lines are dormant the majority of the time, allowing both policies to perform favorably. Additionally, RD performs consistently across all L2 sizes we study, thus it scales well. Our results indicate that the most cost-effective design would be to construct the L2 cache out of high threshold transistors, and only keep the L1 caches with low latency high leakage components. As L2 sizes scale, ideally L1 cache performance and power would remain independent. Unfortunately, the simple policy update window parameter relies on the number of clock cycles: memory behavior for the L1 changes due to different L2 latencies for different sizes. Fig. 16 shows how numbers of drowsy accesses with the simple policy can differ by up to 50% from the largest L2 to the smallest. Drowsy access rates improve for some benchmarks and degrade for others. For example, increasing cache size from 256KB to 2MB reduces drowsy accesses by 47% for art, but a larger cache increases drowsy accesses by 66% for vortex. In contrast, we find RD drowsy accesses to change by less than 1%, thus application performance for L1 will be consistent across changes in L2 size. Furthermore, the RD scheme retains leakage savings for the L1 as L2 scales from 256KB to 2MB.

The increases in L2 cache size result in increases in the number of drowsy accesses, since reuse distances increase. This affects both the simple and RD policies, but the RD policy ensures that leakage remains controlled by capping the number of awake lines. Note that increasing cache size improves IPC, which results in faster running times and a net reduction in leakage (since applications finish faster). Increased cache size does not yield significant increases in leakage, though, since the majority of cache lines are already drowsy.

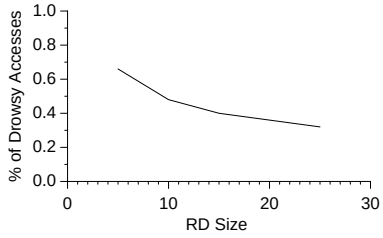


Fig. 12. Drowsy Accesses Normalized to an RD of One

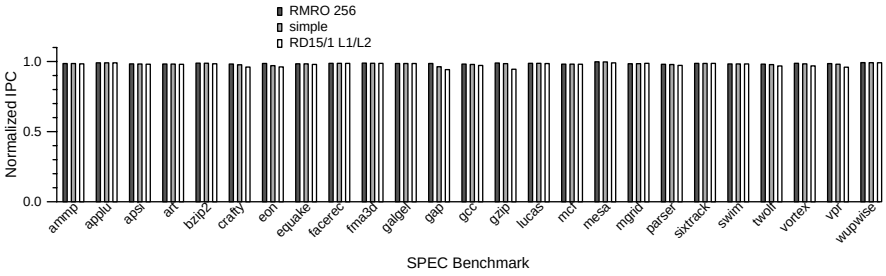


Fig. 13. Drowsy IPCs for 512KB L2

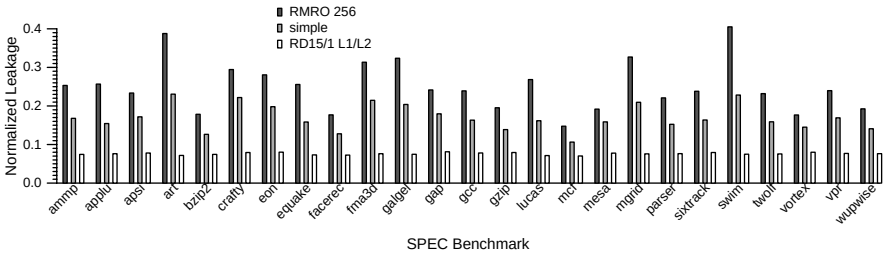


Fig. 14. Drowsy DL1 Leakage for 512KB L2

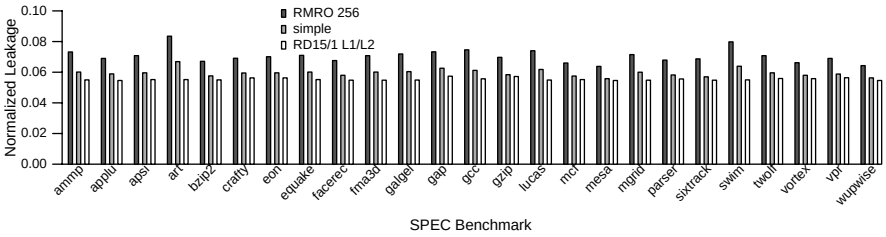


Fig. 15. Drowsy 512KB L2 Leakage

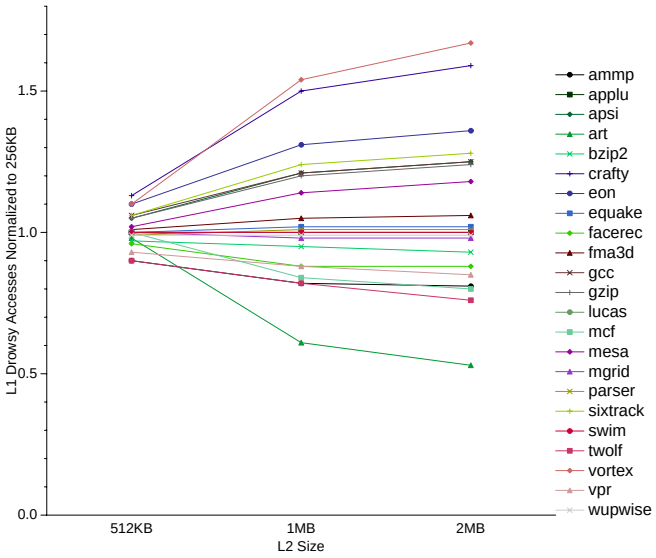
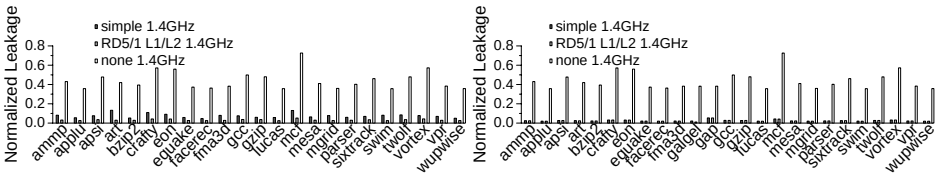


Fig. 16. Drowsy Accesses for Simple Policy as L2 Scales

One advantage of RD is that the power envelope is guaranteed not to be exceeded. The power saved from reducing the leakage can be used for increasing the frequency without regard to potentially having too many cache lines awake at the higher frequency, thereby keeping the original power envelope. The frequency was increased from 500MHz to 1400MHz, while the total power remained constant. Table 4 outlines the power consumption parameters of a an Alpha 21264 processor running at 500MHz on 90nm technology. The net power for the application decreases since the application completes faster. Fig. 17 shows the leakage improvement of a 1.4GHz processor compared to a 500MHz processor with non-drowsy caches. HotLeakage does not model frequency, thus leakage of the 1.4GHz processor has been scaled to yield an approximate representation for the 500MHz processor. Performance improves, but power remains the same, and net energy consumed is reduced. By finely controlling leakage via a stringent drowsy policy, the designer can channel power savings into increasing clock frequency and improving performance. The RD policy retains performance as frequency scales from 500MHz to 1.4GHz. Number of drowsy accesses remains constant in both L1 and L2. In contrast, with the simple policy, number of drowsy accesses increases with increasing frequency, depending on the application.

5 Conclusions

We investigate power reduction techniques for embedded and multi-level cache systems. For embedded systems, we adapt techniques developed to improve cache performance, using them to address both dynamic and leakage power. We revisit



(a) DL1 Leakage for 2MB L2 (b) 2MB L2 Leakage

Fig. 17. Leakage Normalized to 500MHz Non-Drowsy Caches

multiple-access caches in the arena of embedded systems, finding that they generally achieve hit rates equal to larger direct mapped caches while a) reducing static power consumption compared to direct mapped caches, and b) reducing dynamic power consumption compared to normal associative caches. We employ multiple-access region caches with drowsy wordlines to realize further reductions in both dynamic and static power.

With respect to drowsy caching, a simple three- or five-entry Reuse Distance buffer maintaining a few awake (recently accessed) lines performs as well as more complex policies. Our RD drowsy mechanism is easy to implement, scales well with different cache sizes and with number of caches, and enables finer control of power and performance tradeoffs than other published drowsy policies.

Results for most competing drowsy caching solutions are highly dependent on update window sizes. These execution-window based solutions generally employ different intervals for different types of caches. Performance and power properties of all such policies are intimately tied to CPU speed, which means that intervals must be tuned for every microarchitectural configuration (and could be tuned for expected workloads on these different configurations). In contrast, behavioral properties of the RD drowsy mechanism depend only on workload access patterns. Combining multiple-access "pseudo-associativity" with region caching and our RD drowsy policy reduces total power consumption by 16% on average compared to a baseline direct mapped cache with a simple drowsy policy for embedded systems. This savings comes with less than 1% change in IPC. Compared to a direct mapped, non-drowsy region caching scheme, we remain within 1.2% of IPC while realizing power reductions of 65%. We also apply the RD policy to high performance systems with multiple levels of cache hierarchy. We find the L1, drowsy prediction rate is consistently high regardless of changes to the L2. Having an upper bound allows one to divert energy saved from reduced leakage to higher processor frequency in high performance systems. The RD policy provides similar performance (within 1%) to the simple and RMRO policies, while incurring 55% and 70% less leakage, respectively.

A future feature could dynamically change reuse distance sizes depending on device operating mode. Often, handheld electronics have three main modes of operation. A high performance mode, with intensive computation, where the reuse distance is larger or infinite (no drowsy); a standard usage scenario, where reuse distance is optimized for a balance between speed and power; and a third mode, akin to standby, where the device is not being actively used. Such a

standby mode of operation would only need a small reuse distance for significant reductions in leakage with no degradation in quality of service.

Although process technology could potentially solve the leakage problem, future research will investigate steering data to dumb caches that don't require drowsy circuitry (statically assigned high and low threshold cache banks, e.g.) depending on cache access patterns. Well managed drowsy caches will be important to a range of CMP systems, and thus we are beginning to study combinations of energy saving approaches to memory design within that arena. For shared cache resources within a CMP, drowsy policies relying on specified windows of instruction execution become more difficult to apply, making the CPU-agnostic RD mechanism more attractive. In addition, we believe our RD drowsy mechanism to be particularly well suited to asynchronous systems.

References

1. A. Agarwal and S. Pudar. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *Proc. 20th IEEE/ACM International Symposium on Computer Architecture*, pages 169–178, May 1993.
2. D. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proc. IEEE/ACM 32nd International Symposium on Microarchitecture*, pages 248–259, Nov. 1999.
3. D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. 27th IEEE/ACM International Symposium on Computer Architecture*, pages 83–94, 2000.
4. B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *Proc. 2nd IEEE Symposium on High Performance Computer Architecture*, pages 244–253, Feb. 1996.
5. K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proc. 29th IEEE/ACM International Symposium on Computer Architecture*, pages 147–157, May 2002.
6. M. Geiger, S. McKee, and G. Tyson. Beyond basic region caching: Specializing cache structures for high performance and energy conservation. In *Proc. 1st International Conference on High Performance Embedded Architectures and Compilers*, pages 102–115, Nov. 2005.
7. M. Geiger, S. McKee, and G. Tyson. Drowsy region-based caches: Minimizing both dynamic and static power dissipation. In *Proc. ACM Computing Frontiers Conference*, pages 378–384, May 2005.
8. K. Ghose and M. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 70–75, Aug. 1999.
9. M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. IEEE 4th Workshop on Workload Characterization*, pages 3–14, Dec. 2001.
10. K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 273–275, Aug. 1999.

11. K. Inoue, V. Moshnyaga, and K. Murakami. Trends in high-performance, low-power cache memory architectures. *IEICE Transactions on Electronics*, E85-C(2):303–314, Feb. 2002.
12. M. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 143–148, Aug. 97.
13. S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proc. 28th IEEE/ACM International Symposium on Computer Architecture*, pages 240–251, June 2001.
14. N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and microarchitectural techniques for reducing cache leakage power. *IEEE Transactions on VLSI*, 12(2):167–184, Feb. 2004.
15. J. Kin, M. Gupta, and W. Mangione-Smith. Filtering memory references to increase energy efficiency. *IEEE Transactions on Computers*, 49(1):1–15, Jan. 2000.
16. H. Lee. *Improving Energy and Performance of Data Cache Architectures by Exploiting Memory Reference Characteristics*. PhD thesis, University of Michigan, 2001.
17. H. Lee, M. Smelyanski, C. Newburn, and G. Tyson. Stack value file: Custom microarchitecture for the stack. In *Proc. 7th IEEE Symposium on High Performance Computer Architecture*, pages 5–14, Jan. 2001.
18. H. Lee and G. Tyson. Region-based caching: An energy-delay efficient memory architecture for embedded processors. In *Proc. 4th ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 120–127, Nov. 2000.
19. S. Petit, J. Sahuquillo, J. Such, and D. Kaeli. Exploiting temporal locality in drowsy cache policies. In *Proc. ACM Computing Frontiers Conference*, pages 371–377, May 2005.
20. M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 90–95, July 2000.
21. A. Seznec. A case for two-way skewed-associative cache. In *Proc. 20th IEEE/ACM International Symposium on Computer Architecture*, May 1993.
22. T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proc. 10th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, Oct. 2002.
23. P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power, and area model. Technical Report WRL-2001-2, Compaq Western Research Lab, Aug. 2001.
24. Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia Department of Computer Science, Mar. 2003.