

Leveraging High Performance Data Cache Techniques to Save Power in Embedded Systems

Major Bhadauria¹, Sally A. McKee¹, Karan Singh¹, Gary S. Tyson²

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. Voltage scaling reduces leakage power for cache lines unlikely to be referenced soon. Partitioning reduces dynamic power via smaller, specialized structures. We combine approaches, adding a voltage scaling design providing finer control of power budgets. This delivers good performance and low power, consuming 34% of the power of previous designs.

1 Introduction

Power consumption is a first order design criteria for most embedded systems. This is particularly true for newer high-end embedded systems designed for handheld devices such as PDAs, GPS navigators, media players, and portable gaming consoles. These systems not only have faster processor frequencies, but they also place a greater burden on the memory system. This in turn requires large caches both to reduce access latency, and (just as importantly) to avoid large energy costs of off-chip memory accesses when a miss occurs. These larger caches consume a greater portion of the die area and account for a larger percentage of total system power.

Previous research has targeted both static (leakage) power and dynamic switching power. Leakage power can be reduced by reducing the cache size, either by making a cache smaller, or by shutting off portions of it. As an alternative to shutting off parts of the cache, it is possible to reduce the operating voltage for those parts, which reduces the leakage current but increases access latency. Such drowsy caches [6] usually reduce voltage on all cache lines periodically, “waking up” (bringing back to normal voltage) lines as they are accessed. Infrequently accessed lines remain in drowsy mode for long periods.

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. This enables a partitioning strategy to be used to reduce switching

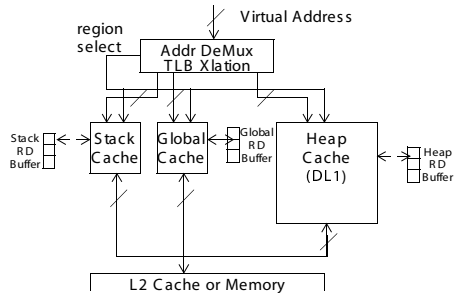


Fig. 1. Organization of Region-Based L1 Caches

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 [20, 18, 19, 7] partition the data cache into separate stack, global and heap caches (as in Figure 1) that retain data from the corresponding memory regions. The heap cache serves as a general L1 Dcache for data not falling in other regions. This partitioning strategy uses a simple address decoder (using only a few address bits) to identify the region and route the reference to the appropriate cache. Switching power goes down, since the region caches are smaller (particularly the stack and global regions).

Our research builds on both the drowsy cache and the region caching work. We develop a new region cache model that maintains original hit rates while reducing leakage via smaller multi-access column associative [1] (CA) and MRU [12] caches. These structures exhibit excellent implementation cost, performance, and energy tradeoffs for the target applications. We also 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 being accessed, keeping a limited (potentially configurable) number of lines awake. Figure 1 illustrates our organization (RD buffers are much smaller than depicted). The RD buffer records IDs corresponding to cache lines being kept awake. 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. A buffer of N entries only needs N counters of $\log_2 N$ bits, since they are incremented every memory access (and not every cycle, as in some previous drowsy policies).

By adapting techniques originally proposed to improve cache *performance*, we develop robust cache organizations to conserve energy in high performance embedded systems. On the MiBench suite, our multiple-access associative RD drowsy policy delivers IPCs of more complicated drowsy designs while using 16% less total power via our multiple-access associative RD policy. Compared to non-drowsy designs, we reduce power by 65% on average, while remaining within 1.2% of the original mean IPC.

2 Related Work

Inoue et al. [13] 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.

2.1 Partitioned Hierarchies

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 [14, 9, 27] and filter caches [17]. These small structures reduce average access energy for hits (i.e., accesses with high temporal locality) at the expense of increased access latency for misses (which increases average L1 latency). Horizontal partitioning reduces dynamic power by a different means: directing accesses at the same hierarchy level to different structures or substructures, as in cache subbanking [9, 27], allows smaller structures. Specialized loop caches [10] and scratchpad memories [29, 21] 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 [20, 18, 19, 7] better exploits temporal and spatial locality behavior of the different data *regions*. This can improve hit rates and cut dynamic and static energy consumption. The Bell Labs C Machine [5] has a separate stack hardware structure, and thus constitutes an early example of a region-based cache.

2.2 Drowsy and Decay Caches

Turning off dead portions of cache after writing dirty lines back to the next level of memory helps control leakage energy. Such *decay caches* [23, 15] often trade performance for reduced power consumption. By predicting dead lines, Kaxiras et al. [15] 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 decreasing supply voltage of the wordlines via dynamic voltage scaling. 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 [16]. 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. [6] investigate *simple* and *noaccess* strategies to keep most

lines drowsy and avoid frequent state changes. The simple policy sets all cache lines to drowsy after a preset number of clock cycles, waking up individual lines as they are accessed; the noaccess policy sets all lines not accessed within the interval to drowsy. The simple policy offers aggressive leakage reduction, whereas the noaccess policy is intended to yield better performance by keeping actively used lines awake. In practice, the policies exhibit minor performance differences, and thus most drowsy structures use the simple policy for its ease of implementation [6]. Geiger et al. add drowsiness to their heap cache [8] and explore a separate, smaller, non-drowsy *hot heap cache* to hold lines with highest temporal locality [7]. Since the stack and global caches service most references, an aggressive drowsy policy in the heap cache has negligible effects on performance. Petit et al. [22] propose a *Reuse Most Recently used On* (RMRO) drowsy cache that behaves much like a noaccess policy adapted for set associativity, using update intervals to calculate how often a set is accessed. The MRU way remains awake during the interval, but other ways of infrequently used sets are turned off. RMRO is more sophisticated than the simple policy, but requires additional hardware per set and uses dynamic power every cycle. None of these policies can place a hard limit on total cache leakage.

2.3 Multiple-Access Caches

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 [12], Half-and-Half [28], skew associative [25], and predictive sequential associative [4] caches. Hash-rehash caches swap lines on rehash hits to move the most recently accessed line to the original lookup location. CA caches avoid thrashing lines between the 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 to predict the way to access first in a two-way associative cache reduces power consumption with minimal performance effects. Their ease of implementation and good performance characteristics make CA and MRU way-predictive associative structures attractive choices for low power hierarchies; other multiple-access designs focus on performance over energy savings.

Process	70nm
Operating Voltage	1.0V
Operating Temperature	65° 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)
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

3 Experimental Setup

We use SimpleScalar [3] with HotLeakage [30] (which models drowsy caching) for the ARM [24] ISA to model the 32 benchmarks from the MiBench suite [11]. This suite represents a range of commercial embedded applications from the automotive, industrial, consumer, office, network, security, and telecom sectors. Our simulator has been adapted for region caching [7], and we incorporate column associativity and MRU way prediction, along with our RD drowsy policy. The processor is single-issue and in-order with a typical five-stage pipeline, as in Table 1. We calculate static power consumption via HotLeakage, and use both CACTI 3.2 [26] and Wattch [2] to calculate dynamic power consumption. We calculate memory latency for single accesses via CACTI. To establish a valid baseline comparison with Geiger et al.’s previous region caching work [8] we use their cache configuration parameters: separate, single-ported 32KB L1 instruction and data caches with 32B lines. Table 2 gives full details of our region based caching configurations.

The main forms of power dissipation are static current leakage and dynamic switching of transistors transitioning among different operating modes. HotLeakage calculates static cache leakage as a function of the process technology and operating temperature (since leakage doubles with every 10° C increase). Operating temperature is 65° C, which is suitable for embedded systems for personal electronics. Dynamic power consumption for typical logic circuits is $\frac{1}{2}CV^2f$, a function of the frequency, capacitance and operating voltage. We choose CACTI

	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

over Wattch [2] to model dynamic power for all our caches since the former more accurately calculates the number of sense amplifiers in associative caches. We convert CACTI energy numbers to power values based on architecture parameters in Table 2. We use CACTI values to accurately model dynamic power and combine these with HotLeakage’s leakage numbers to compute total power. Since CACTI reports energy and uses 0.9V for 70nm while Wattch and HotLeakage use power and 1.0V for that process technology, we convert our numbers using the appropriate frequency (1.7GHz) scaling $P=E/t$ where $E=\frac{1}{2}C \times V^2$ for $V=0.9$ to $V=1$. These conversions ensure accuracy of numbers being added from different modeling tools.

Figure 2 illustrates the organization of a Reuse Distance drowsy mechanism. The RD policy 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 which buffer entry is the LRU. The RD circuitry never dictates what lines to awaken, but only which 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 $\log_2 N$ bits that are updated every memory access (and not every cycle). Assuming a reuse distance size of eight and 1024 cache lines (as for a 32KB 32-Byte line baseline cache), storing the awake cacheline IDs and current LRU counts requires 104 bits ($[\log_2 1024 + \log_2 8 \text{ bits}] * 8$), of which only a single buffer’s count value (three bits) would be 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.

4 Evaluation

We compare our baseline region caches to organizations in which the heap cache is replaced with a multiple-access cache (CA or MRU) of half the size. Access patterns for the stack and global cache 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 drowsy policies. Keeping all

lines drowsy incurs an extra cycle access penalty that lowers IPC by up to 10% for some applications. If performance is not crucial, using always drowsy caches is an attractive design point. We find that the noaccess drowsy policy always uses slightly more power than the simple policy, although it yields slightly higher average IPC by about 0.6%. Given the complexity of implementation, the low payoff in terms of performance, and the lack of energy savings, we use the simple policy in our remaining comparisons.

We find that 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, 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. Figure 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 cache³. 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. [6] and Petit et al. [22], suffer 20% more leakage energy. Thus the simple drowsy policy is dependent on update window size, whereas the RD policy is independent of any update interval (and CPU frequency). This means it 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. Figure 4 shows that for all benchmarks, on average, 66% of the intervals

³ These values were found to be optimal for simple drowsy policies and region caches [8].

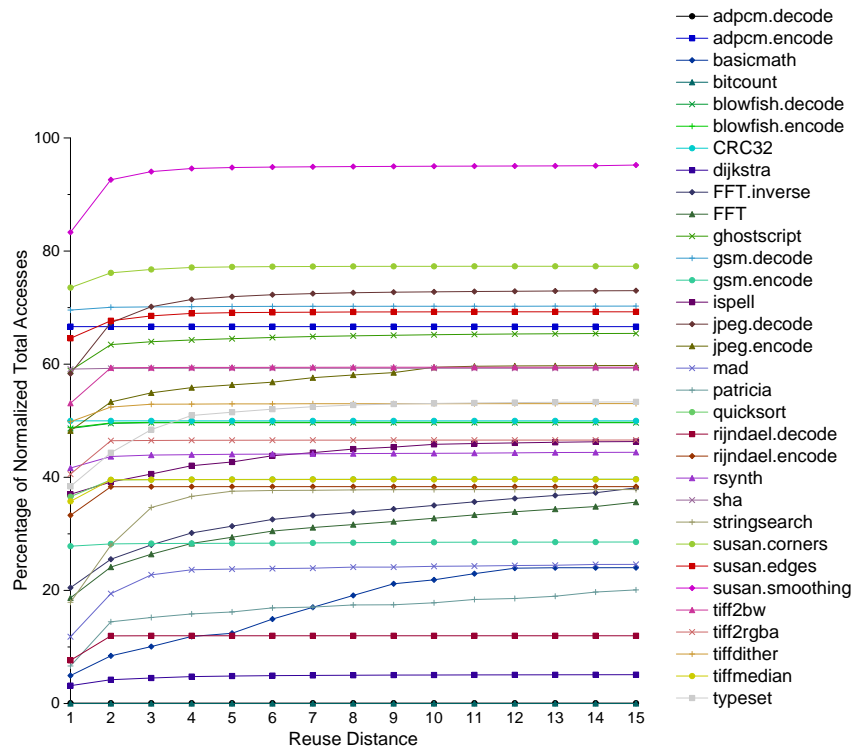


Fig. 3. Reuse Distances for Last 15 Cache Lines

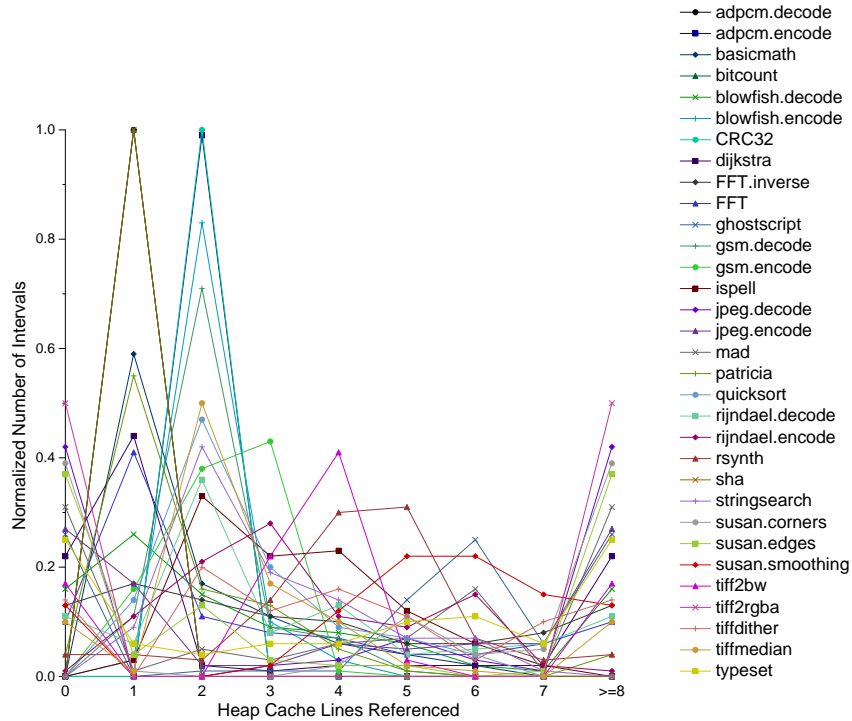


Fig. 4. Number of Heap Cache Lines Accessed During *simple* Intervals (512 Cycles)

access fewer than four lines. These data are normalized to the total number 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 strategy consumes less dynamic power than a conventional two-way set associative cache that charges two wordlines simultaneously (which means that 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. Figure 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.

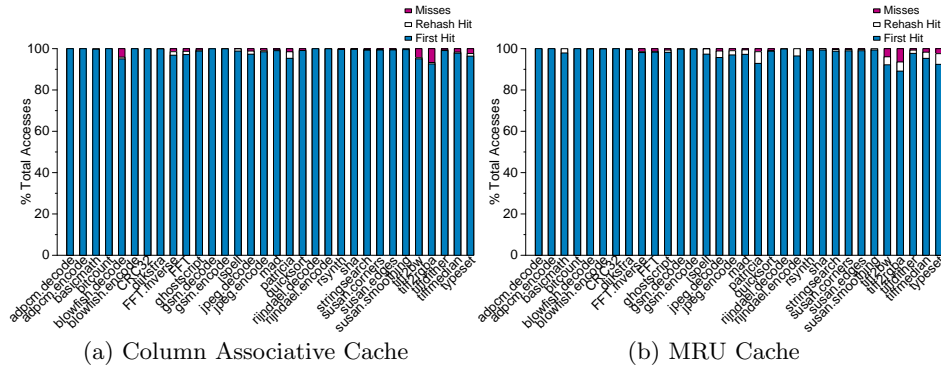


Fig. 5. Heap Accesses Broken Down by Category

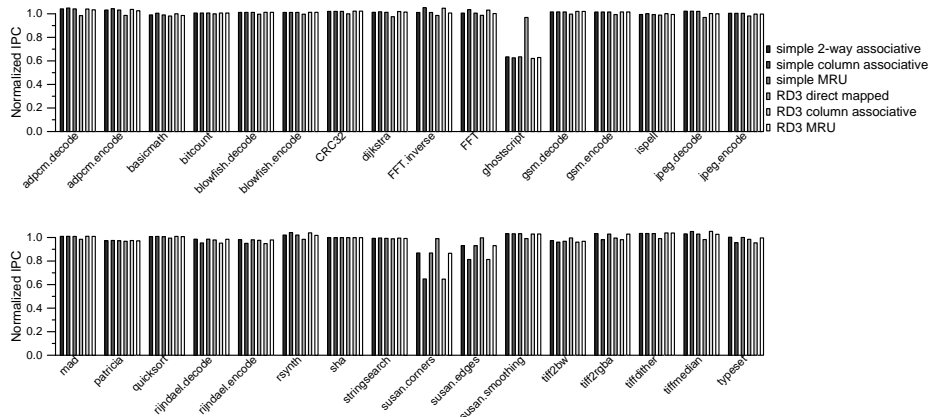


Fig. 6. IPCs Normalized to *simple* Drowsy Direct Mapped Region Caches

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 of 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 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. Figure 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.

4.1 Sustaining Performance

Figure 6 graphs IPCs relative to direct mapped caches with the best update windows from Geiger et al. [8]. In the majority of benchmarks, two-way set associative, CA, and MRU caches match or exceed the performance of their direct mapped counterparts, but they 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 benchmarks, which have a higher hit rate with the associative configuration, but a lower IPC than the direct-mapped cache configuration.⁴ Overall, IPCs are within 1% of the best baseline case, as with the simple and noaccess results of Flautner et al. [6]. These differences are small enough to be attributed to modeling error.

4.2 Reducing Dynamic Power

Figure 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. Figure 7 illustrates this for CA and MRU organizations: rehash checks and way mispredictions increase power consumption for some scenarios.

4.3 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 Figure 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. The simple and RD policies exhibit similar savings, but the RD mechanism is easier to implement and tune, without requiring window size calibration for each workload. Ideally, update window size for simple drowsy policies and RD buffer sizes would be software-configurable for optimal power-performance tradeoffs among different workloads. No single setting will always yield best results.

The effects of heap cache dynamic energy savings shown in Figure 7 represent a small contribution to the total L1 Dcache power consumption shown in Figure 9. Implementing drowsy policies across all memory regions plays a

⁴ We find these results to be anomalous: porting to the Alpha ISA and running on a single-issue, in-order, architecturally similar Alpha simulator results in the way-associative caches having higher IPCs for these benchmarks.

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 negligible performance reduction of less than 1.2%.

5 Conclusions

We adapt techniques developed to improve cache performance, using them to address both dynamic and leakage power. We revisit multiple-access caches within the arena of high-performance embedded systems, finding that they generally achieve equal hit rates 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 (RD) buffer that maintains a small number of awake (recently accessed) cache lines performs as well as the more complex policies used in most studies in the literature. Our RD drowsy mechanism is easy to implement, scales well with different cache sizes, scales well with number of caches, and offers finer control of power management 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. 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%.

Future research will investigate combining an L2 decay cache with drowsy L1 data caches. 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. 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.
3. D. Burger and T. Austin. The simplescalar toolset, version 2.0. Technical Report 1342, University of Wisconsin, June 1997.
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. D. Ditzel and H. McLellan. Register allocation for free: The c machine stack cache. In *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 48–56, Mar. 1982.
6. 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.
7. 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*, Nov. 2005.
8. 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.
9. 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.
10. A. Gordon-Ross and F. Vahid. Frequent loop detection using efficient nonintrusive on-chip hardware. *IEEE Transactions on Computers*, 54(10):1203–1215, Oct. 2005.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.

17. 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.
18. H. Lee. *Improving Energy and Performance of Data Cache Architectures by Exploiting Memory Reference Characteristics*. PhD thesis, University of Michigan, 2001.
19. 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.
20. 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.
21. M. Ohmacht, R. A. Bergamaschi, S. Bhattacharya, A. Gara, M. E. Giampapa, B. Gopalsamy, R. A. Haring, D. Hoenicke, D. J. Krolak, J. A. Marcella, B. J. Nathanson, V. Salapura, and M. E. Wazlowski. Blue Gene/L compute chip: Memory and ethernet subsystem. *IBM Journal of Research and Development*, 49(2-3):255–264, 2005.
22. 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.
23. 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.
24. S. Santhanam. StrongARM SA110: A 160MHz 32b 0.5W CMOS ARM processor. In *Proc. 8th HotChips Symposium on High Performance Chips*, pages 119–130, Aug. 1996.
25. A. Sez nec. A case for two-way skewed-associative cache. In *Proc. 20th IEEE/ACM International Symposium on Computer Architecture*, May 1993.
26. 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.
27. Su and A. Despain. Cache designs for energy efficiency. In *Proc. 28th Annual Hawaii International Conference on System Sciences*, pages 306–315, Jan. 1995.
28. K. Theobald, H. Hum, and G. Gao. A design framework for hybrid-access caches. In *Proc. 1st IEEE Symposium on High Performance Computer Architecture*, pages 144–153, Jan. 1995.
29. M. Verma, L. Wehmeyer, and P. Marwedel. Cache-aware scratchpad allocation algorithm. In *Proc. ACM/IEEE Design, Automation and Test in Europe Conference and Exposition*, pages 21264–21270, Feb. 2004.
30. 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.