

# A Precisely Tunable Drowsy Cache Management Mechanism

Major Bhadauria, Sally A. McKee, Karan Singh

Cornell University

Ithaca, NY 14853

{major | sam | karan}@csl.cornell.edu

Gary Tyson

Florida State University

Tallahassee, FL 32306

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, leakage current becomes an increasingly important component in overall power. Since most cache lines remain idle most of the time, drowsy caching techniques that reduce voltages on selected lines can reduce this leakage power. Most drowsy caching policies in the literature update drowsy/non-drowsy state after a given execution window: e.g., the widely used *simple policy* puts all caches lines to sleep after a specified number of clock cycles. Such methods are inherently architecture-specific. We instead introduce a drowsy caching mechanism that has intuitive appeal in its exploitation of 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 better with cache size and hierarchy depth. This mechanism keeps only the last  $N$  recently used lines awake, where  $N$  is configurable. This *reuse distance* (RD) policy not only delivers equivalent or better results at lower implementation complexity and execution overhead, but it allows more precise control than previous policies with respect to enforcing a strict power budget. We apply the RD policy to the SPEC benchmarks, comparing it to the simple policy for a range of cache hierarchies. For a 32KB L1 data cache backed by a 512KB L2, on average RD uses less than 44% of the power of the simple policy for L1 and about 93% for L2. RD and simple deliver 97.6% and 98.8%, respectively, of the IPC of the same non-drowsy configuration, thus RD delivers almost identical performance at substantial power savings.

## 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 power increase stems from exponential growth in leakage due to the decreasing transistor voltage thresholds that accompany shrinking feature sizes. The bottom line is that large caches cause significant current leakage: Kim et al. find cache leakage to constitute up to 70% of total power in a 70nm process [15].

The number of transistors leaking current can be reduced by turning caches off or decreasing their sizes, both of which are likely to incur concomitant decreases in performance. Since most cache lines remain idle most of the time, drowsy caching techniques [7] that reduce voltages on selected lines can help reduce leakage power. Some mechanism must implement a policy to decide when to turn lines off and on. Two main policies have been studied in the literature: the *simple policy* indiscriminately puts all caches 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 window of cycles (also clock-frequency dependent). Simple has been shown to perform almost identically to the more sophisticated noaccess, and thus is most often used in drowsy cache organizations [7]. Petit et. al [19] introduce a drowsy policy tailored for way-associative caches, where ways are put to sleep depending on their

usage (this policy attempts to reduce leakage similarly to Albonesi’s [2] work on selective cache ways for dynamic power reduction). All these policies rely on counters that use elapsed clock cycles to determine when cache lines should be put to sleep. Such methods are inherently architecture-specific, and prevent prediction of leakage savings before individual benchmark evaluation, which we find to be a drawback.

We improve on previous work by introducing a new drowsy caching mechanism that has intuitive appeal in its exploitation of 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 better with cache size and hierarchy depth. This mechanism keeps only the last  $N$  recently used lines awake, where  $N$  is configurable. This *reuse distance* (RD) policy not only delivers equivalent or better results at lower implementation complexity and execution overhead, but it allows more precise control than previous policies with respect to enforcing a strict power budget. This yields consistent leakage savings across benchmarks.

We apply the RD policy to 25 SPEC CPU 2000 benchmarks, comparing with the simple policy for 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 94.5% for 512KB L2 caches, respectively. Note that the latter represents a percentage of a much larger number, and thus represents significant energy savings. As L2 grows relative to L1, our approach maintains both performance and power savings by keeping fewer L2 lines awake. Finally, our policy supports larger overall cache sizes and/or higher clock rates, delivering better performance while adhering to a strict power budget. We thus propose a data cache organization that better matches reference behavior and provides much finer grained control for dynamic power management methods.

## 2 Related Work

As cache systems have evolved to reduce memory access energy, two methods of integrating multiple caches have been developed, horizontal and vertical partitioning schemes. Vertical partitioning schemes like line buffers [13, 9, 24] and filter caches [17] place a new cache between the processor and L1. These additional caches are smaller than the L1, thereby reducing access power. Line or block buffering retains the most recently accessed cache line(s). When the next reference occurs, the buffer is checked before accessing the main cache.

If the access hits the buffer, the main cache need not be activated. These small structures reduce average access energy for references with high temporal locality, but misses to these L0 structures increase average L1 latency, which can lead to significant increases in execution time for many applications. Victim buffers [12] and L3 caches represent other vertical partitionings.

Horizontal partitioning is applied to a single level of the cache hierarchy by splitting a given cache into multiple, smaller structures. Dynamic power is reduced when references are serviced by one of the smaller caches or cache sub-banks [9, 24]. Different types of references exhibit different spatial and temporal locality properties, and breaking the L1 data cache into separate, smaller structures for stack, global, and heap accesses [18, 8] improves hit rates while reducing both dynamic and static energy consumption. An early example of such *region-based caching* is the Bell Labs C Machine [6] with its separate hardware structure for stack data.

Leakage energy may be controlled by turning off portions of cache holding dead data after writing dirty lines back to the next level of memory. Such *decay caches* [20, 14] trade performance for power reduction. Kaxiras et al. [14] predict dead lines via adaptive hardware schemes based on competitive algorithms, realizing reductions in leakage energy by factors of four-five with little or no impact on performance.

Even when cache lines are not dead, they may be read infrequently. Placing idle cache lines in a dormant state-preserving (*drowsy*) condition reduces static power consumption by decreasing the supply voltage of the word-lines using dynamic voltage scaling. A drowsy word-line consists of a drowsy bit, voltage control mechanism, and a word-line gating circuit [16]. The drowsy bit controls switching between the high and low voltages, and the word-line gate protects the data from being accessed in drowsy mode, since the drowsy line must be brought back to its original voltage before being loaded in the sense-amplifiers. Extensive performance and power overheads result from repeatedly waking and putting to sleep the same lines, hence Flautner et al. [7] explore strategies for activating drowsy lines (we compare against their *simple* policy here).

Petit et al. [19] propose RMRO drowsy caches, which use a noaccess-type policy adapted for set associativity. They record information per set and use update intervals to calculate how often a set is accessed before turning it off. The most recently used way remains awake. RMRO is more sophisticated than the simple policy, but requires additional hardware per set and uses dynamic power every cycle. Like the simple and noaccess policies, RMRO cannot place a hard limit on total cache leakage.

|                              |  |
|------------------------------|--|
| 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<br>1-cycle Access<br>32B Lines                      |
| L1 Dcache                    | 32KB 4-Way Associative<br>1-cycle Access<br>32B Lines                      |
| L2 Cache                     | 256KB/512KB/1MB/2MB<br>4-way Associative<br>4/10/27/32 cycles<br>32B Lines |
| Main Memory                  | 97 cycles  |

**Table 1.** Architectural Parameters

Associative caches deliver about the same performance as direct mapped caches twice their size, and thus have potential to save leakage energy via smaller structures. *Multiple-access* or *Hybrid Access* caches try to deliver the performance of more highly associative organizations at lower associativity. These cache designs allow data to reside at more places in the cache by performing another lookup at a *reshashed* address location on a miss, trading lower energy costs for a slight increase in complexity and in access time for additional lookups. Examples include the hash rehash, column associative [1], MRU [10], Half-and-Half [25], skew associative [21], and predictive sequential associative [4] caches.

### 3 Experimental Setup

We evaluate drowsy caching policies via SimPoints [22] for 25 SPEC CPU 2000 applications with reference inputs on a validated model of a four-issue Alpha architecture [5]. We modify this to model both Flautner et al.’s simple drowsy caching mechanism [7] and our Reuse Distance (RD) drowsy mechanism. To model power consumption, we use HotLeakage [26], which incorporates the Wattch [3] dynamic power framework. Table 1 gives base architectural parameters of our Alpha 212264 model: to establish a valid baseline comparison with Flautner et al.’s work we use their cache configuration 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-

|                                  |        |
|----------------------------------|--------|
| 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 2.** Benchmark-Independent Power Parameters (Watts) for Frequency Scaling

associative and either 256KB, 512KB, 1MB or 2MB in size (with appropriate memory latencies calculated via CACTI [23]). We model drowsy L1 data and L2 caches. Switching lines from high to low (sleep mode) or low to high (wake-up mode) voltages incurs a one-cycle transition penalty. We keep tag lines awake, so only hits to drowsy cache lines suffer the 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 C, which is typical of a CPU, and a 70nm process technology, which is sufficiently state-of-the-art to support future cores with significantly larger L2 caches.

Table 2 reports power parameters extracted via Wattch for modest clock frequencies of 500MHz and 1.4GHz: the table values are constant across all benchmarks. RD5 indicates that we keep five lines awake (here in the L1 Dcache); RD1 indicates that we keep a single line awake (in the L2 cache). Using the RD5 and RD1 policies reduces leakage by 92% and 94% for our L1 and L2 caches, respectively. In generating these numbers, we account for power consumed by cache circuitry, including the mechanisms to implement RD drowsy caching.

Figure 1 illustrates the organization of a Reuse Distance drowsy mechanism. The RD policy tracks lines be-

|   | 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                                  |

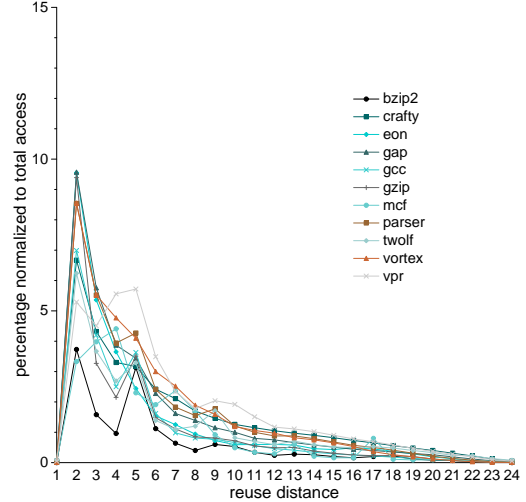
**Figure 1.** Organization of Drowsy LRU Structure for an RD of Eight

ing 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, and they are updated on each 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.

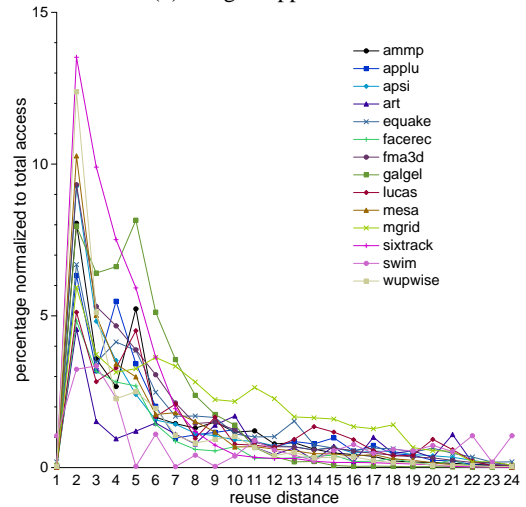
## 4 Evaluation

We apply the simple and RD policies to the L1 data and L2 caches, comparing their leakage power and performance to a non-drowsy cache architecture. Dynamic power consumption is not affected by changing drowsy policies, nor by the changes in IPC (from scaling clock frequencies, e.g.), since it is independent of running time. We assume increases in net dynamic power due to clock switching for a lower IPC are negligible.

The simple policy uses a 4000-cycle execution window before all cache lines expire and are turned off; previous research finds this to be the optimal window size [7]. Figure 2 and Figure 3 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-



(a) Integer Applications

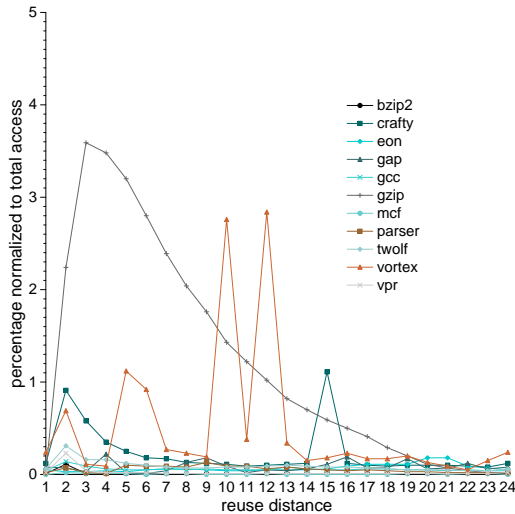


(b) Floating Point Applications

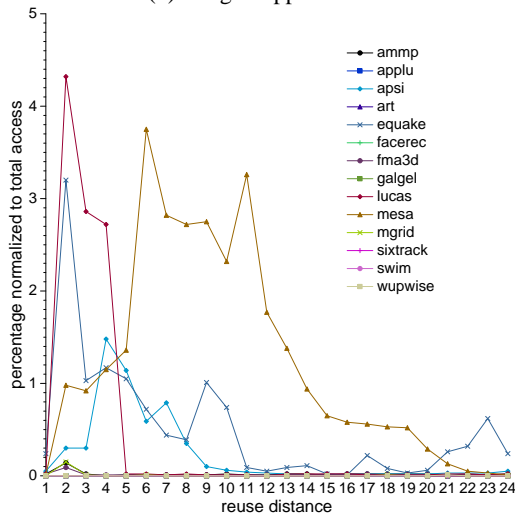
**Figure 2.** L1 Temporal Locality

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 five lines awake in the L1 and one line in the L2 (i.e., keeping the most recently accessed line awake). Note that a drowsy access increases L1 latency significantly (by 50%) compared to a drowsy L2 access.



(a) Integer Applications

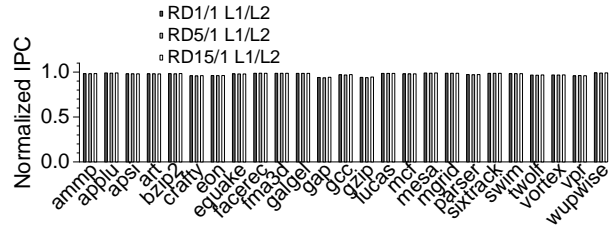


(b) Floating Point Applications

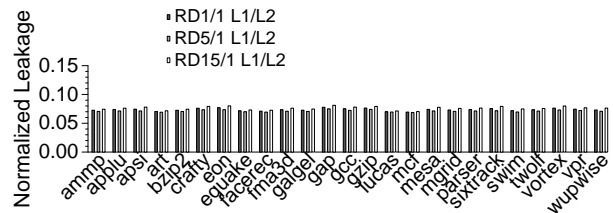
**Figure 3.** L2 Temporal Locality

We therefore spend the majority of our power budget masking performance degradation at the L1 level first. To illustrate the effect of changing RD sizes, we show IPC and static power consumption normalized to non-drowsy caches for RD L1/L2 buffer combinations of 1/1, 5/1 and 15/1 in Figure 4 through Figure 6. IPC degradation for all cases is negligible (less than 1%), but decreases in IPC cause some benchmarks to use more energy with RDs of 1/1 over RDs of 5/1. Differences between the RD policies are accentuated in Figure 7, which shows numbers of drowsy accesses.

Figure 8 through Figure 19 compare IPCs and leakage for RD versus simple as L2 size grows from 256KB to 2MB (all results normalized to non-drowsy caches). For smaller L2 sizes, both deliver significant leakage sav-



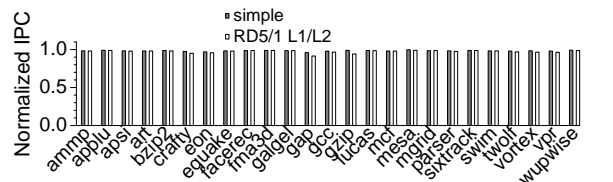
**Figure 4.** Drowsy RD IPCs for 512KB L2



**Figure 5.** Drowsy RD DL1 for 512KB L2



**Figure 6.** Drowsy RD 512KB L2



**Figure 8.** Drowsy IPCs for 256KB L2

ings with almost indiscernible performance degradation (less than 3%). RD delivers substantial savings at the L1 level (56.1% over simple), since the number of awake lines is capped at five at any time. RD achieves lower power consumption than simple with the L2 cache, but the improvement is not as high as in 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.

As L2 sizes scale, ideally L1 cache performance and power would remain independent. Unfortunately, the simple policy execution window parameter relies on the number of clock cycles, and thus memory behavior for the L1 changes due to different L2 latencies for different

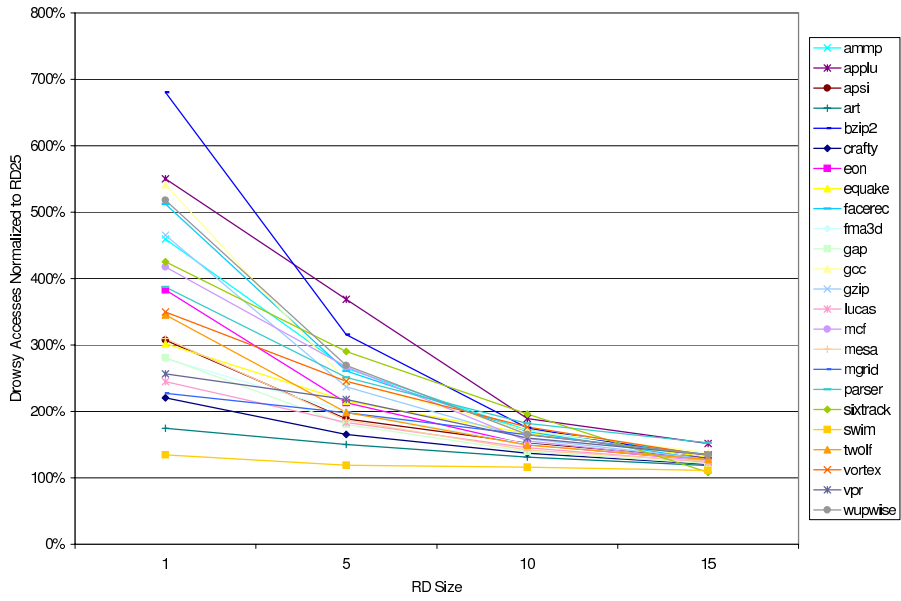


Figure 7. Drowsy Accesses Normalized to RD25

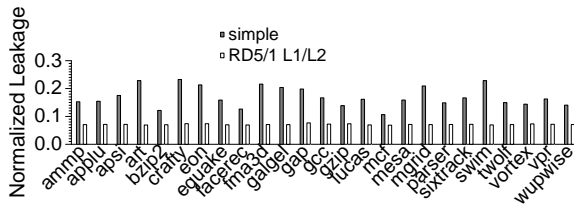


Figure 9. Drowsy DL1 Leakage for 256KB L2

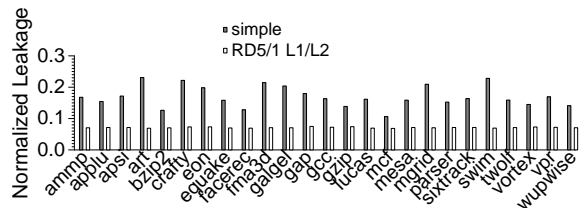


Figure 12. Drowsy DL1 Leakage for 512KB L2

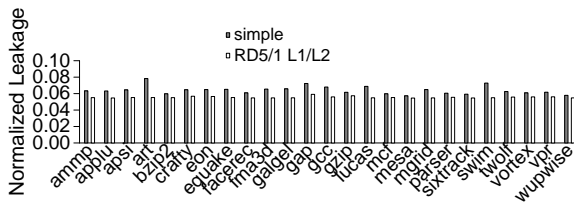


Figure 10. Drowsy 256KB L2 Leakage

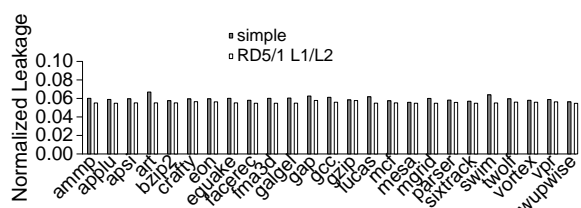


Figure 13. Drowsy 512KB L2 Leakage

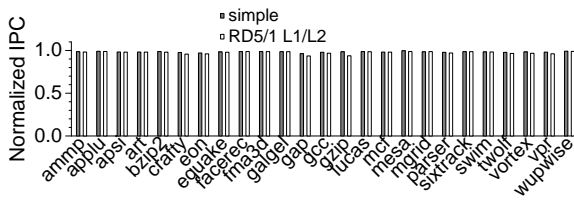


Figure 11. Drowsy IPCs for 512KB L2

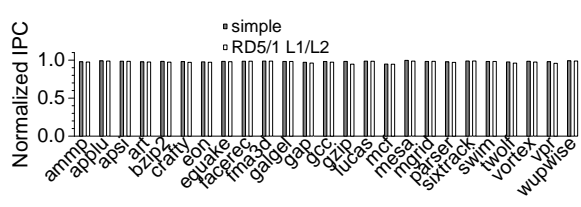


Figure 14. Drowsy IPCs for 1MB L2

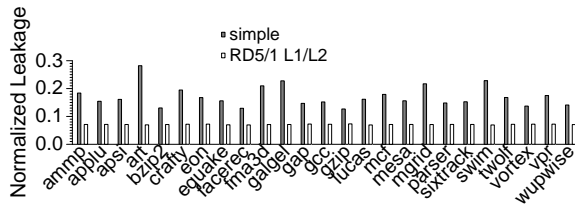


Figure 15. Drowsy DL1 Leakage for 1MB L2

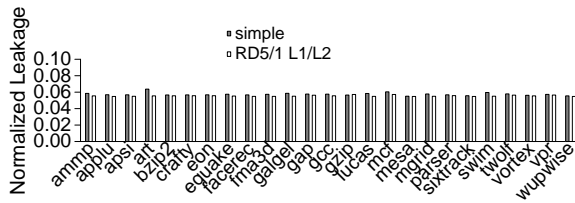


Figure 16. Drowsy 1MB L2 Leakage

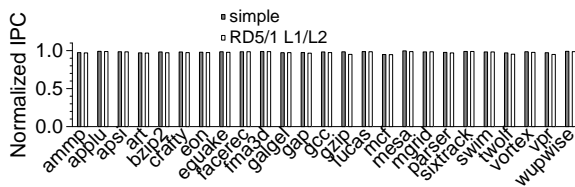


Figure 17. Drowsy IPCs for 2MB L2

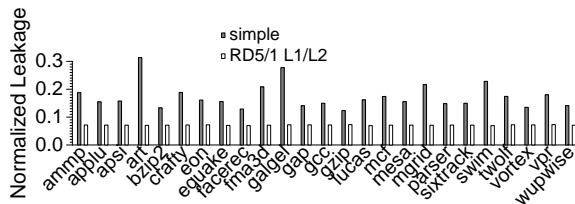


Figure 18. Drowsy DL1 Leakage for 2MB L2

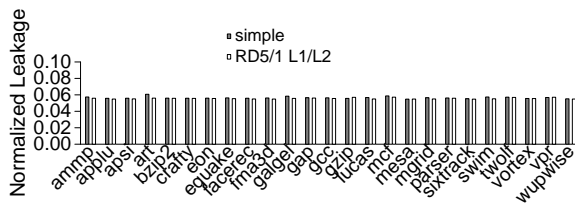


Figure 19. Drowsy 2MB L2 Leakage

cache sizes. Figure 20 shows how the number 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 as cache size increases. For example, increasing cache size from 256KB to 2MB reduces the number of drowsy accesses by 47% for art, but a larger cache increases the number of drowsy accesses by 66% for vortex.

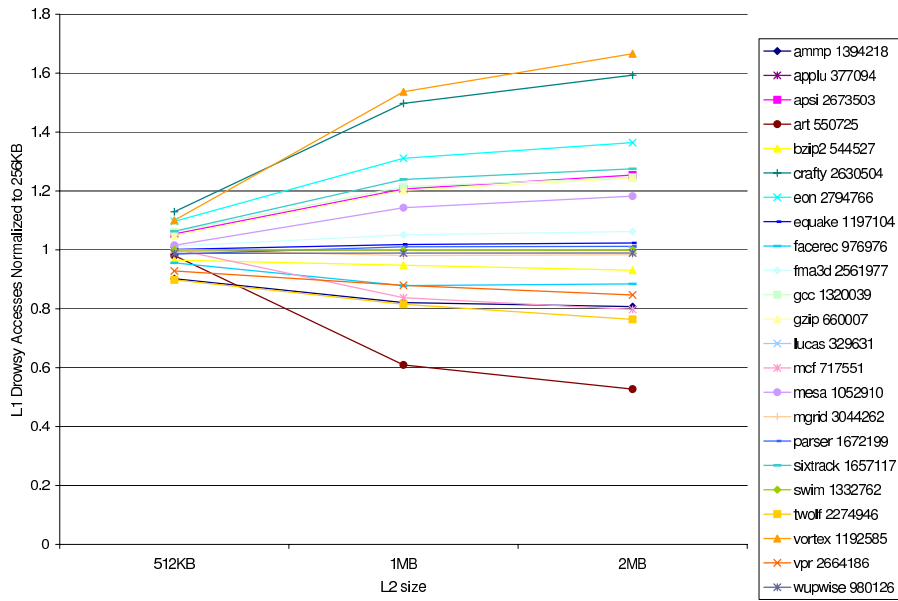
RD performance does not change with scaling, as illustrated by Figure 21: the number of drowsy accesses across all benchmarks changes by less than 1%. This ensures that application performance for L1 will be consistent across changes in L2 cache size. Furthermore, the RD scheme retains leakage savings 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.

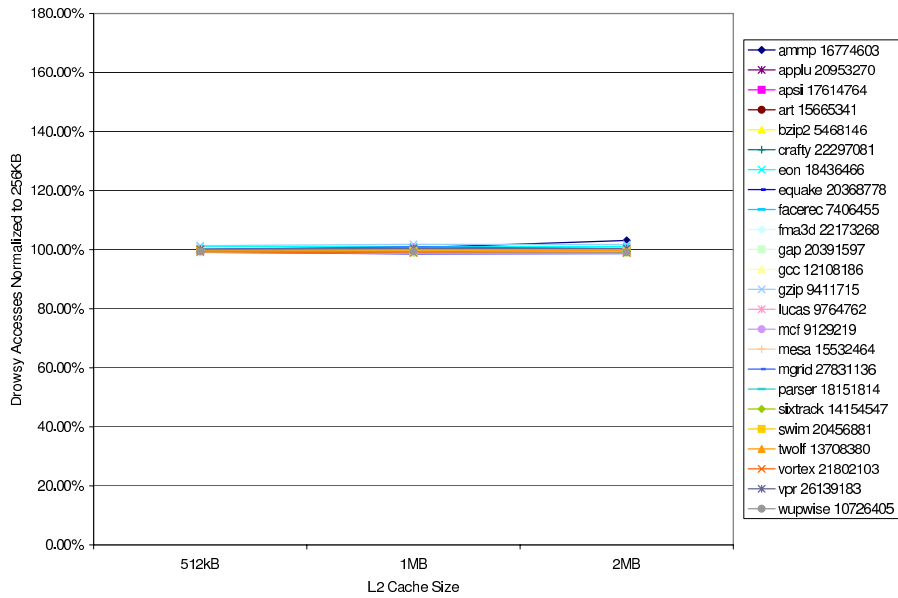
We choose a low RD value to improve leakage savings, but increasing the RD buffer improves hit rates and reduces IPC degradation. Having an RD of 50/1 (L1/L2) improves performance to within 0.5% of simple performance, while delivering significant reductions in leakage, as shown in Table 3. For some benchmarks, such as mgrid, the RD50/1 policy can deliver lower power and better performance than the simple policy: for that particular benchmark, performance is 0.2% better than simple, and L1/L2 leakage is approximately 58% and 9% less than simple, respectively.

The RMRO policy is similar to the noaccess policy modified for the associative case, where if a cache way has been accessed within a time window it remains awake, otherwise it's turned off. If more than two ways have been accessed, only the two MRU ways are kept awake. We implement the RMRO policy using the optimal window of 1024 found by Petit et al. [19] Comparing RMRO1024, RMRO256, simple policy, and an RD of 50/1 indicates negligible performance differences and significant power savings for RD as detailed in Table 3.

Figure 22 shows the improvement in leakage of a 1.4GHz processor compared to a 500MHz processor with non-drowsy caches. Since HotLeakage does not model frequency, the leakage of the 1.4GHz processor has been scaled to yield an accurate representation for



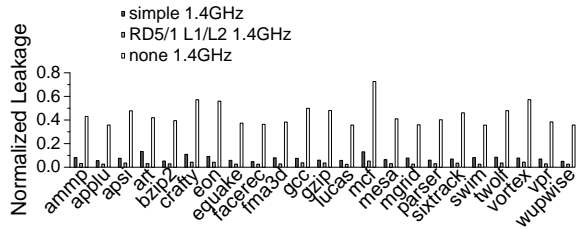
**Figure 20.** Drowsy Accesses for Simple Policy as L2 Scales



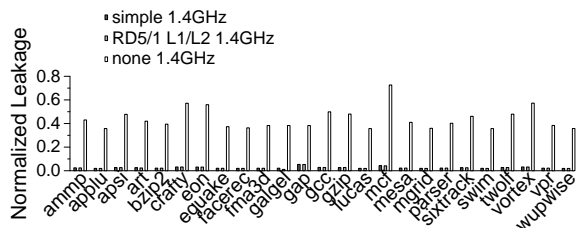
**Figure 21.** Drowsy Accesses for RD5 Policy as L2 Scales

|            | RMRO256 | RMRO1024 | RD50  | RD100 |
|------------|---------|----------|-------|-------|
| IPC        | 99.9%   | 100.2%   | 99.6% | 99.7% |
| L1 Leakage | 87.5%   | 146.4%   | 55.6% | 66.2% |
| L2 Leakage | 108.0%  | 118.3%   | 92.9% | 93.0% |

**Table 3.** Results Normalized to Simple 4096



**Figure 22.** DL1 Leakage for 2MB L2 Normalized to 500MHz Non-Drowsy Caches



**Figure 23.** 2MB L2 Leakage Normalized to 500MHz Non-Drowsy Caches

the 500MHz processor. Performance improves, while power remains the same, and the 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. The number of drowsy accesses remains constant in both L1 and L2. In contrast, with the simple policy, the number of drowsy accesses increases with the increase in frequency, depending on the benchmark.

## 5 Conclusions

We have introduced a drowsy mechanism that is impervious to changes in memory latency and provides upper and lower bounds on expected power consumption. When this reuse distance (RD) policy is applied to 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 sys-

tems. The RD policy provides similar performance to the simple policy, while incurring 56% less leakage.

Future research will investigate combining an L2 decay cache with a drowsy L1 data cache. We believe well managed drowsy caches will be important to CMP systems, and are beginning to study various 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. Finally, 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. Albonese. 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] R. Desikan, D. Burger, and S. Keckler. Measuring experimental error in multiprocessor simulation. In *Proc. 28th IEEE/ACM International Symposium on Computer Architecture*, pages 266–277, June 2001.
- [6] 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.
- [7] 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.
- [8] 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.
- [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] 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] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990.
- [13] 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.
- [14] 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.
- [15] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches: Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proc. IEEE/ACM 36th International Symposium on Microarchitecture*, pages 219–230, Nov. 2002.
- [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 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. Sez nec. 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] Su and A. Despain. Cache designs for energy efficiency. In *Proc. 28th Annual Hawaii International Conference on System Sciences*, pages 306–315, Jan. 1995.
- [25] 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.
- [26] 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.