HIGH PERFORMANCE TECHNIQUES FOR REDUCING CACHE POWER

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Major Balram Bhadauria

May 2008

© 2008 Major Balram Bhadauria
ALL RIGHTS RESERVED

ABSTRACT

Minimizing power consumption continues to grow as a critical design issue for many platforms, from embedded systems to chip multiprocessors (CMP) to ultra scale parallel systems. Embedded systems, like their desktop counterparts, have migrated to a multi-core architecture. Power is a first-order design component in the embedded domain, and advances in process technology have led to a cascading effect. Chiefly, decreasing feature sizes have led to lower voltage thresholds (to retain performance), thereby resulting in exponential increases in leakage. Leakage has now become a fundamental design concern with respect to total power budget. This issue has been postponed by using different SRAM cell designs on current process technologies.

Two approaches have been proposed to reduce the power requirements of cache memories: voltage scaling (or "drowsiness") and partitioning. Voltage scaling targets leakage current by reducing the voltage to cache lines unlikely to be referenced soon. Partitioning targets dynamic switching power by splitting the cache into smaller structures, either banks or regions. We combine the best of these two approaches by developing a new region cache organization. We add a new voltage scaling design that enables finer control of cache lines than previous voltage scaling policies. We evaluation this new organization on embedded and high performance architectures, finding it provides similar high performance and much lower power consumption than previously published low-power cache designs.

BIOGRAPHICAL SKETCH

Major Bhadauria received his B.S. in Electrical and Computer Engineering from University of Toronto in May 2004. He is currently a Ph.D. student at Cornell's Computer Systems Laboratory within the School of Electrical and Computer Engineering. He is partly supported by an NSERC doctoral fellowship from 2008-2010.

To the women in my life for their enduring support and wisdom.

ACKNOWLEDGEMENTS

I would like to acknowledge several individuals that were instrumental in the completion of this thesis. First, I thank my advisor Sally McKee for being the best advisor I could hope for, for teaching me how to do research in computer architecture and for guiding my academic career. In the years I have been a graduate student, she has made me truly feel like a family member within our research group. This thesis would not be possible without her dedication and hard work. I would also like to thank Vince Weaver, Karan Singh and Brian White for their suggestions and motivation.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . vi
List of Tables . . . vii
List of Figures . . . viii

1 Introduction **1**

2 Related Work **6**
2.1 Partitioned Hierarchies . . . 6
2.2 Drowsy and Decay Caches . . . 7
2.3 Multiple-Access Caches . . . 10

3 Motivation **11**

4 Experimental Setup **15**

5 Evaluation **20**
5.1 High Performance . . . 20
5.2 Embedded Architectures . . . 29
5.2.1 Sustaining Performance . . . 31
5.2.2 Reducing Dynamic Power . . . 33
5.2.3 Reducing Leakage Current . . . 34

6 Conclusions and Future Work **36**

Bibliography **38**

LIST OF TABLES

4.1 Architectural Parameters 16
4.2 Benchmark-Independent Power Parameters (Watts) for Frequency Scaling 16
4.3 Baseline Processor Configuration 18
4.4 Region Drowsy Cache Configuration Parameters 18

LIST OF FIGURES

1.1 Organization of RD-Based L1 Region Caches . . . 3

2.1 Example of a Vertical Cache Hierarchy . . . 8
2.2 Common Example of Horizontal Cache Partitions . . . 8

3.1 L1 Temporal Locality for Reuse Distances up to 24 (INT left, FP right) . . . 12
3.2 L2 Temporal Locality for Reuse Distances up to 24 (INT left, FP right) . . . 12
3.3 Reuse Distances for Last 15 Cache Lines . . . 13
3.4 Number of Heap Cache Lines Accessed During *simple* Intervals (512 Cycles) 14

5.1 Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches) . . . 21
5.2 Drowsy L1 for 512KB L2 (Normalized to Non-Drowsy Caches) . . . 21
5.3 Drowsy L2 for 512KB L2 (Normalized to Non-Drowsy Caches) . . . 21
5.4 Drowsy Accesses Normalized to RD25 . . . 22
5.5 Drowsy IPCs for 256KB L2 (Normalized to Non-Drowsy Caches) . . . 22
5.6 Drowsy DL1 Leakage for 256KB L2 (Normalized to Non-Drowsy Caches) . . 23
5.7 Drowsy DL2 Leakage for 256KB L2 (Normalized to Non-Drowsy Caches) . . 23
5.8 Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches) . . . 23
5.9 Drowsy DL1 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches) . . 24
5.10 Drowsy DL2 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches) . . 24
5.11 Drowsy IPCs for 1MB L2 (Normalized to Non-Drowsy Caches) . . . 24
5.12 Drowsy DL1 Leakage for 1MB L2 (Normalized to Non-Drowsy Caches) . . . 24
5.13 Drowsy DL2 Leakage for 1MB L2 (Normalized to Non-Drowsy Caches) . . . 25
5.14 Drowsy IPCs for 2MB L2 (Normalized to Non-Drowsy Caches) . . . 25
5.15 Drowsy DL1 Leakage for 2MB L2 (Normalized to Non-Drowsy Caches) . . . 25
5.16 Drowsy DL2 Leakage for 2MB L2 (Normalized to Non-Drowsy Caches) . . . 25
5.17 Drowsy Accesses For Simple Policy as L2 Scales . . . 26
5.18 Drowsy Accesses For RD5 Policy as L2 Scales . . . 27
5.19 Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches) . . . 28
5.20 Drowsy DL1 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches) . . 28
5.21 Drowsy DL2 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches) . . 28
5.22 DL1 Leakage for 2MB L2 Normalized to 500MHz Non-Drowsy Caches . . . 29
5.23 DL2 Leakage for 2MB L2 Normalized to 500MHz Non-Drowsy Caches . . . 29
5.24 Heap Accesses Broken Down by Category (CA Cache on Left, MRU Cache on Right) . . . 31
5.25 IPCs Normalized to *simple* Drowsy Direct Mapped Region Caches . . . 32
5.26 Dynamic Power Consumption of Associative Heap Caches Normalized to a Direct Mapped Heap Cache . . . 32
5.27 Static Power Consumption of Different Drowsy Policies with Direct Mapped Region Caches Normalized to Non-Drowsy Direct Mapped Region Caches . . 32
5.28 Total Power of Drowsy Policies (for All Region Caches) and Heap Cache Organizations Normalized to Direct Mapped Region Caches . . . 33

CHAPTER 1

INTRODUCTION

Although the density of transistors per silicon area continues to almost double every 18 months as first noted by Gordon Moore, the frequency at which these transistors operate has reached a plateau. This is attributed to power on chip not increasing with advances in computer architecture and VLSI, because of physical and thermal constraints. Computers are now shifting to multi-core architectures so throughput can continue to increase without increasing power consumption. Since dynamic power consumption is $\frac{1}{2}CV^2f$, halving the frequency and voltage results in quadratic power reductions (but not cubic, since a two-core CMP results in double the capacitance, which negates the savings from halving the frequency). Doubling the cores usually results in doubling of the caches, which accounts for more than half the die area, in an effort to mask the speed difference between memory and the processor. Total power consumption consists of dynamic and static power, which, combined, results in caches consuming 70% or more of the total chip power at the 70nm process technology. Static power consumption exponentially increases with decreasing process technology (due to reductions in voltage threshold and substrate materials to increase speed). Since total available chip power is finite, we investigate methods for maximizing performance while working under a strict power budget, specifically reducing L1 and L2 power consumption so that high performance chips can maintain their high frequency with large L2 caches.

Moore's Law has resulted in commodity low-cost high performance ASICs that can be incorporated into several mobile application areas such as GPS, entertainment systems such as MP3/DVD/game players, business tools such as PDAs, and medical devices. With increasing chip count and advances in process technology, the static power consumption has increased dramatically, leading to shortened operating times of mobile devices. Additionally, these systems not only have faster processor frequencies, but

they also place a greater burden on the memory system. This requires smarter caches, to reduce access latency, and (just as importantly) to avoid large energy costs of off-chip memory accesses when a cache miss occurs [30]. Larger caches, the traditional solution, consume a greater portion of die area and account for a larger percentage of total system power. To alleviate this, we investigate methods of reducing dynamic and static power consumption for high performance systems and embedded architectures.

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 [8] 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 (wake-up or put to sleep). Two main policies have been studied in the literature: the *simple policy* [8] 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* [8] 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 [8]. Petit et. al [24] 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.

When examining previous drowsy policies, their reliance on clock cycles was disconcerting, since we desire solutions that will scale with multiple cache hierarchies and

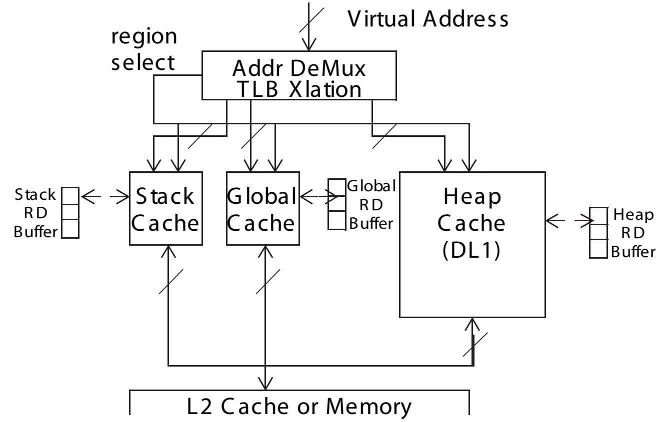

Figure 1.1: Organization of RD-Based L1 Region Caches

multicores, where access latencies are not constant. Unlike the simple policy, we want a policy that explicitly uses cache lines accessed in determining when to turn off cache lines, and we want to remain independent of clock cycles. Preliminary work shows there is a high temporal correlation between recent cache lines, indicating a policy based on reuse distance has negligible performance degradation, while providing guaranteed leakage savings. This reuse distance (RD) policy tracks lines being accessed, keeping a limited number of lines awake.

We improve on previous work by introducing the RD drowsy caching mechanism with its intuitive appeal in its exploitation of temporal locality. It 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 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 address static leakage consumption for the high performance and embedded do-

mains, by applying the RD policy on drowsy caches we incorporate within their respective cache hierarchies. We further reduce power consumption in the embedded domains by leveraging high performance cache access techniques to reduce cache size while retaining cache hit rates. This leads to further dynamic and static power reductions.

We apply the RD policy to 25 SPEC CPU 2000 benchmarks, comparing with the simple policy for a range of hierarchies. Using the cycle-accurate SimpleScalar simulator for the alpha processor, simulating modest clock rates of 0.5-1.5 GHz, we observe overall 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 smaller percentage of the much larger former, 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.

Switching or dynamic power consumption can also be minimized by reducing cache size. 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 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 [22, 20, 21, 9] partition the data cache into separate stack, global and heap caches that retain data from the corresponding memory regions. The heap cache functions as a general L1 data cache (Dcache) for any 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. Since region caches are smaller, particularly the stack and global regions, switching power is reduced. Figure 1.1 illustrates our organization, where we extend the region cache design to incorporate RD buffers (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_2N bits, since they are incremented every memory access (and not every cycle, as in previous drowsy policies) [8].

We develop new region cache models that maintain original hit rates while reducing leakage via smaller multi-access column associative [1] (CA) and MRU [14] caches. These structures exhibit excellent implementation cost, performance, and energy tradeoffs for the target applications [13].

By adapting techniques originally proposed to improve cache *performance*, we develop robust cache organizations to conserve *energy* in high performance and embedded systems. For high performance benchmarks, our policy achieves 56% leakage savings over competing solutions on the Alpha architecture, with a less than 1% variation in IPC. On the MiBench suite, we deliver IPCs of more complicated drowsy designs while using 16% less total power via our multiple-access associative RD drowsy policy. Compared to non-drowsy designs, we achieve power reductions of 65% on average, while remaining within 1.2% of the original mean IPC.

The rest of this thesis addresses previous work in the area of reducing cache power consumption in Chapter 2, details the motivation behind our approach in Chapter 3, presents our simulation infrastructure in Chapter 4, evaluates our own power reduction mechanisms in Chapter 5, and presents our conclusions in Chapter 6.

CHAPTER 2

RELATED WORK

Inoue et al. [15] 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 structures 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 [31]. Figure 2.1 shows a memory hierarchy consisting of three levels of caches, an L1, and a larger L2 cache which is backed by main memory. Vertical caches are generally inclusive of data in smaller caches higher up in the memory hierarchy. This design has several advantages, chiefly performance and power. By initially checking the L1 cache, which is small, the cache lookup can be performed very quickly (target being a single clock cycle). Secondly, due to the nature of the L1 cache's small size, the power consumed on a cache access is significantly less than accessing the L2 cache which is an order of magnitude larger on modern systems. The temporal and spatial locality characterisitics of memory access patterns plays a large role in reducing the number of fetches from one cache to another. The inherent disadvantage of such a hierarchy is the wasted space of having data redundantly in multiple locations and the worst case speed degradation of looking for data in multiple locations consecutively (L1, than L2, etc.) Modern vertical partitionings may create an L0 memory level between the processor and L1, as in line buffers [16, 11, 32] and filter caches [19]. These small structures further reduce average access energy for hits (i.e., accesses with high temporal locality) at the expense of increased access la-

tency 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 [11, 32], allows smaller structures. By dividing a large cache into sub-banks, the performance and power benefits of small caches are achieved with the hit rate and capacity of a large cache, at the expense of added die area overhead. Specialized loop caches [12] and scratchpad memories [34, 23] 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 [22, 20, 21, 9] potentially better exploits the temporal and spatial locality behavior of the different data *regions*. This can improve hit rates, and cut both dynamic and static energy consumption and cache access time since they are smaller structures. The Bell Labs C Machine [7] has a separate hardware structure for stack data, and thus constitutes an early example of a region-based cache. Figure 2.2 shows the common case of the L1 partitioned between instruction and data caches. A memory request can be filled by either of the caches, but never both as the data they cache respecitvely is exclusive (instruction cache only caches program instructions, and data cache only caches program data). Another cache partitioning strategy is to reduce the number of cache ways, which was used by Albonesi [2] for saving power. His research extracts signficant energy reductions from large L2 caches with negligible performance impact, depending on cache associativity and workload.

2.2 Drowsy and Decay Caches

Turning off dead portions of a cache after writing dirty lines back to the next level of memory helps control leakage energy. Such *decay caches* [25, 17] often trade performance for reduced power consumption. Kaxiras et al. [17] reduce L1 leakage energy by

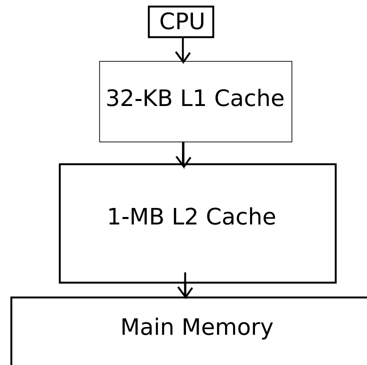

Figure 2.1: Example of a Vertical Cache Hierarchy

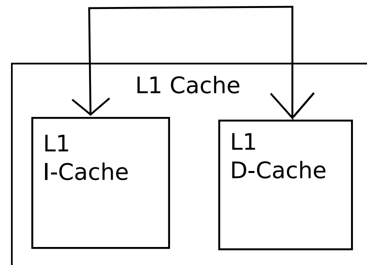

Figure 2.2: Common Example of Horizontal Cache Partitions

up to a factor of five with little impact on performance by predicting dead lines. They add a two-bit counter per line to track 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 time to these lines is increased. Repeatedly changing state incurs high performance and power overheads. Drowsy wordlines consist of a drowsy bit, voltage control mechanism, and wordline gating circuit [18]. 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. [8] 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 [8]. Geiger et al. add drowsiness to their heap cache [10] and explore a separate, smaller, non-drowsy *hot heap cache* to hold lines with highest temporal locality [9]. 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. [24] 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 [14], Half-and-Half [33], skew associative [27], and predictive sequential associative [5] 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.

CHAPTER 3

MOTIVATION

Evaluation of the RD policy is motivated by the temporal reuse property exhibited by cache lines. The simple policy uses a 4000-cycle execution window before all cache lines expire and are turned off; previous research has already found this to be the optimal window size [8]. However, to find an optimal RD size, histograms of the reuse distance are graphed in Figure 3.1 and Figure 3.2, detailing the percentages of total cache accesses having reuse distances from one to 24 cache lines. The reuse distance could be as large as the total number of cache lines (in our configuration 1024 lines for the L1), but that would result in no leakage savings, and temporal locality characteristic hints that it would be unnecessary for a buffer even half that size for optimal performance. The histograms indicate that temporal locality drops significantly after the most recent five unique line references. Temporal locality for the L2 is a 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). Therefore 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 is configurable. 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. Again, 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

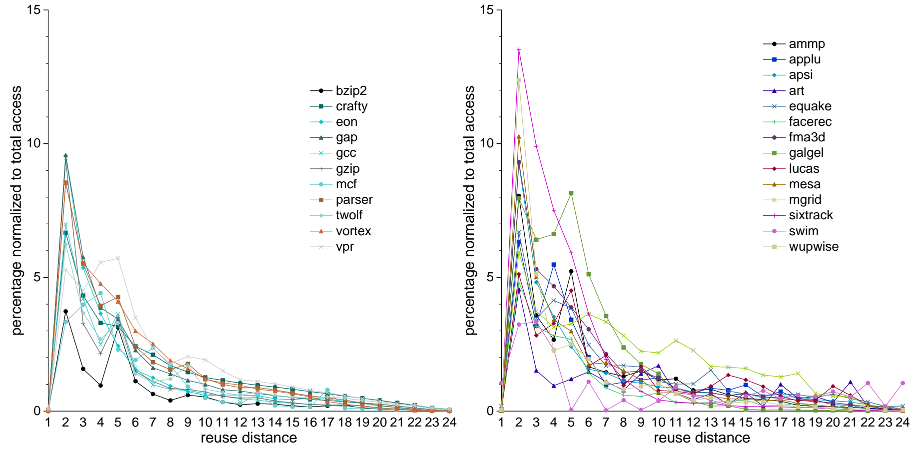

Figure 3.1: L1 Temporal Locality for Reuse Distances up to 24 (INT left, FP right)

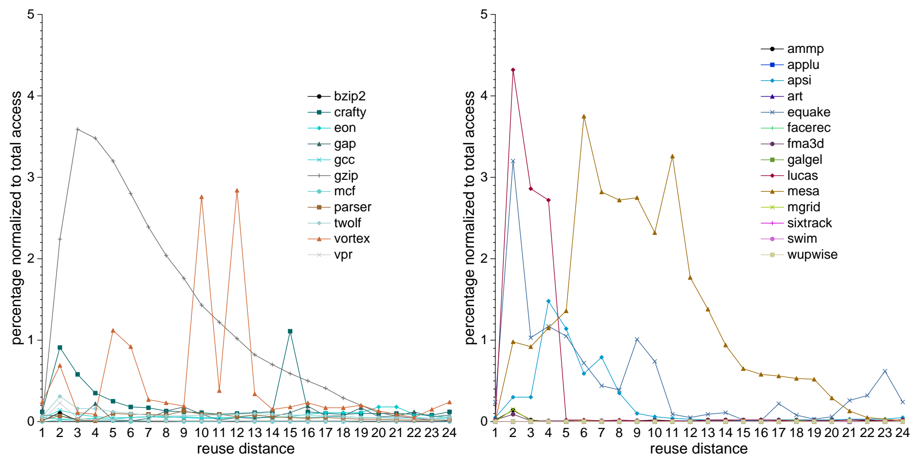

Figure 3.2: L2 Temporal Locality for Reuse Distances up to 24 (INT left, FP right)

equal to the buffer size. Static power overhead is negligible relative to cache sizes.

For the embedded domain, 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 3.3 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

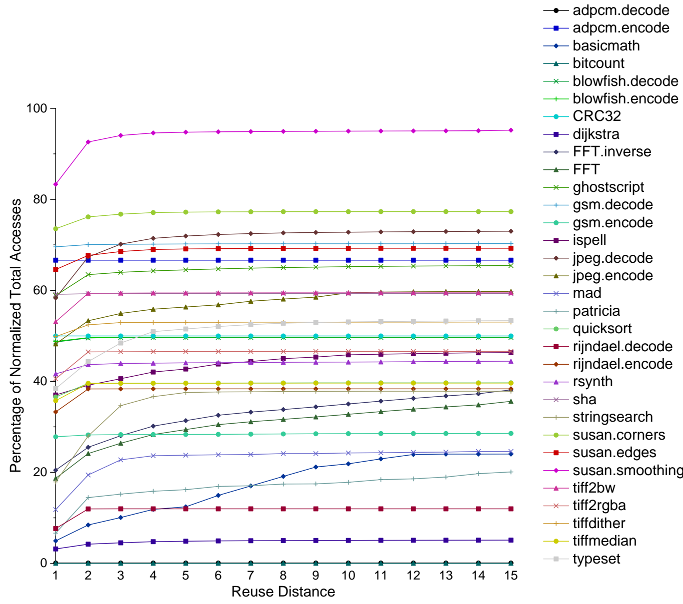

Figure 3.3: Reuse Distances for Last 15 Cache Lines

all heap accesses.

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 3.4 shows that for all benchmarks, on average, 66% of the intervals access fewer than four lines. These data are normalized to the total number of intervals per benchmark so that detail for shorter-running benchmarks is preserved. Benchmarks such as em 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.

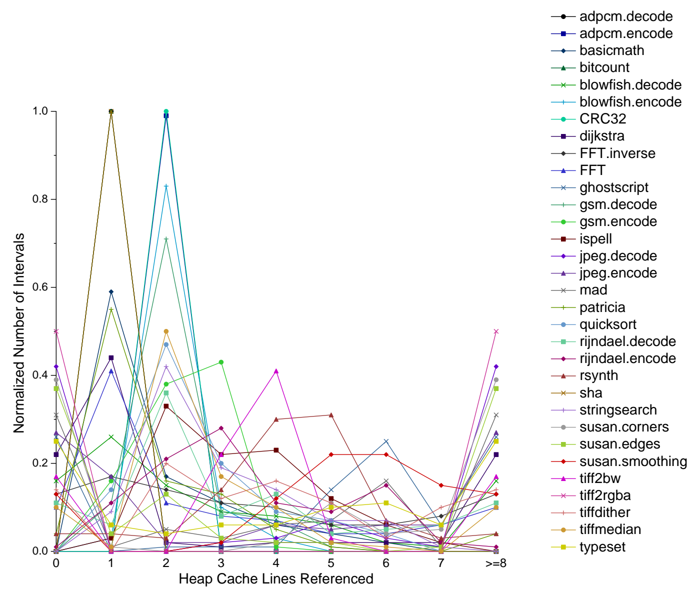

Figure 3.4: Number of Heap Cache Lines Accessed During *simple* Intervals (512 Cycles)

CHAPTER 4

EXPERIMENTAL SETUP

We evaluate drowsy cache policies for the embedded and high performance domains. For the embedded domain we test the MiBench benchmark suite compiled for the ARM ISA and architecture. In the high performance arena, we evaluate the SPEC2000 applications on the Alpha ISA and architecture. We simulate this work using the cycle-accurate SimpleScalar simulator, which we modify to capture power and energy data.

We evaluate drowsy caching policies via SimPoints [28] for 25 SPEC2000 applications with reference inputs on a validated model of a four-issue Alpha architecture [6]. We modify this simulator to model both Flautner et al.'s simple drowsy caching mechanism [8] along with our Reuse Distance (RD) drowsy mechanism. To model power consumption, we use HotLeakage [35], which incorporates the Wattch [3] dynamic power framework. Table 4.1 gives base architectural parameters of our Alpha 21264 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-associative and either 256KB, 512KB, 1MB or 2MB in size (with appropriate memory latencies calculated via CACTI [29]). 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

Table 4.1: Architectural Parameters

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 4.2: Benchmark-Independent Power Parameters (Watts) for Frequency Scaling

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

with significantly larger L2 caches.

Table 4.2 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.

We use SimpleScalar [4] with HotLeakage [35] (which models drowsy caching) for the ARM [26] ISA to model the MiBench suite [13], consisting of 32 benchmarks. 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 [9], 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 4.3. We calculate static power consumption via HotLeakage, and use both CACTI 3.2 [29] and Wattch [3] 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 [10] we use their cache configuration parameters: separate, single-ported 32KB L1 instruction and data caches with 32B lines. Table 4.4 gives full details of our region based caching configurations.

Note that RD circuitry for determining what lines to turn off does not lie in the critical path, so this policy can be applied to any drowsy method without increasing delay. Cache lookups occur as normal, waking up lines as needed, and when the buffer is full, a cache line not currently being used is put to sleep. Based on CACTI access time calculations, it takes one cycle to access the RD LRU registers, and another to increment

Table 4.3: Baseline Processor Configuration

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 4.4: Region Drowsy Cache Configuration Parameters

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

and write them back. This can be done in parallel with the cache lookup. This ensures that the logic for putting cache lines to sleep can keep up with cache requests.

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. 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 over Wattch [3] 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 4.4. 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.

CHAPTER 5

EVALUATION

We evaluate our work under two different domains, high performance and embedded. We first examine applying drowsy caches on a multi-hierarchy cache configuration for the high performance domain. We than investigate the benefits of leveraging multiple techniques to reduce dynamic and static power for the embedded domain where power is of utmost concern.

5.1 High Performance

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, and is not affected by the changes in IPC (from scaling clock frequencies, e.g.) since it is independent of running time. Increase in net dynamic power due to clock switching for a lower IPC is assumed to be negligible.

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. 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 for RD L1/L2 buffer combinations of 1/1, 5/1 and 15/1 in Figure 5.1 through Figure 5.3. 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 5.4, which shows number of drowsy accesses.

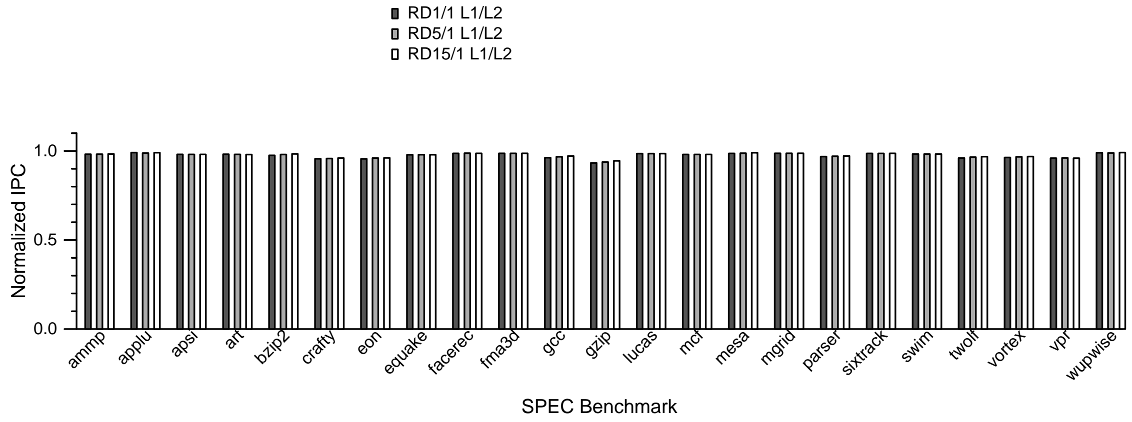

Figure 5.1: Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches)

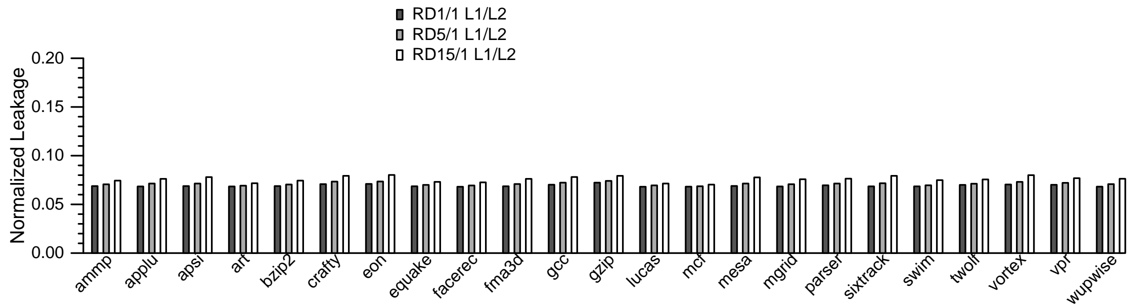

Figure 5.2: Drowsy L1 for 512KB L2 (Normalized to Non-Drowsy Caches)

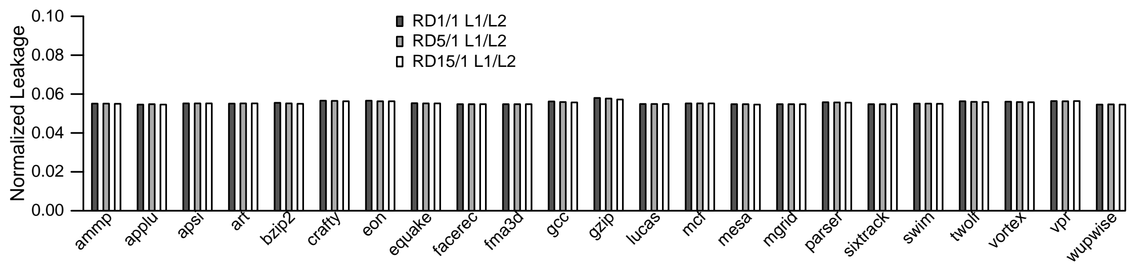

Figure 5.3: Drowsy L2 for 512KB L2 (Normalized to Non-Drowsy Caches)

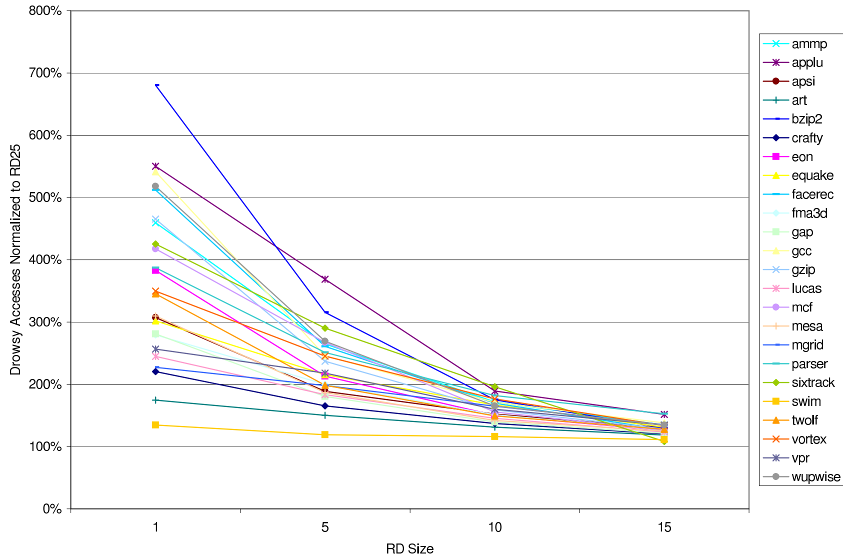

Figure 5.4: Drowsy Accesses Normalized to RD25

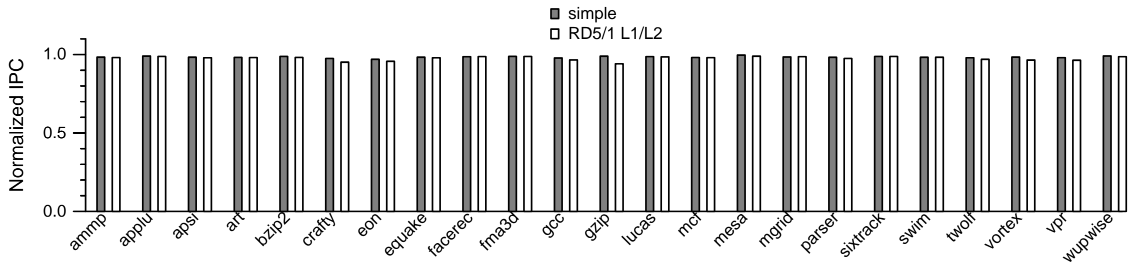

Figure 5.5: Drowsy IPCs for 256KB L2 (Normalized to Non-Drowsy Caches)

Figure 5.5 through Figure 5.16 compare IPCs and leakage for RD versus simple as L2 size grows to 2MB. For smaller L2 sizes, both deliver significant leakage savings 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 up, ideally L1 cache performance and power would remain in-

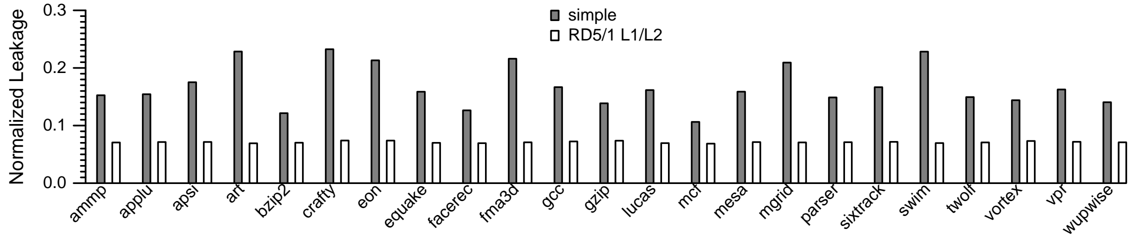

Figure 5.6: Drowsy DL1 Leakage for 256KB L2 (Normalized to Non-Drowsy Caches)

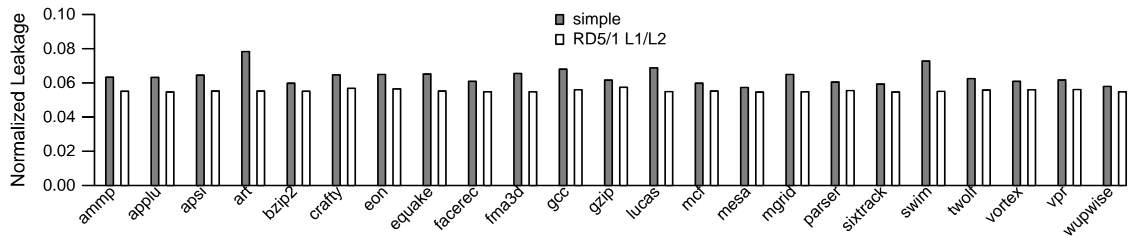

Figure 5.7: Drowsy DL2 Leakage for 256KB L2 (Normalized to Non-Drowsy Caches)

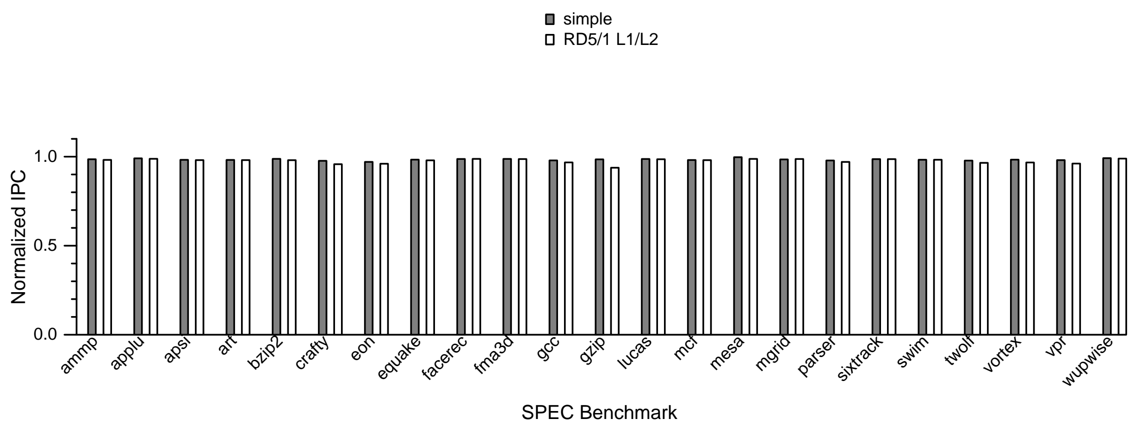

Figure 5.8: Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches)

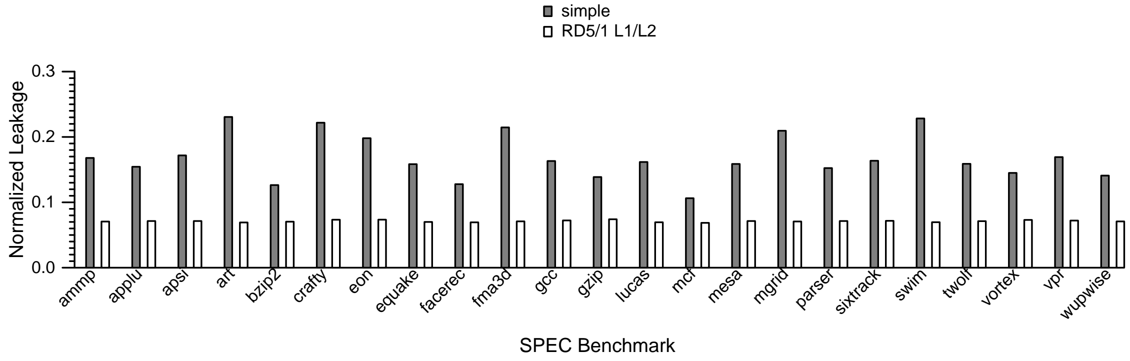

Figure 5.9: Drowsy DL1 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches)

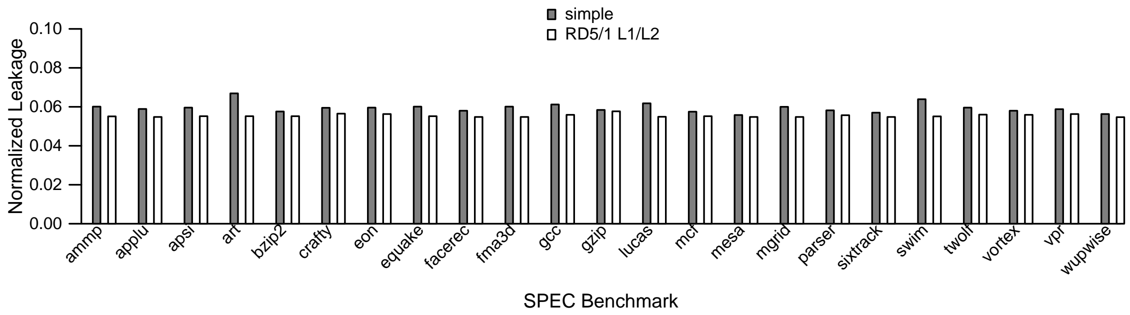

Figure 5.10: Drowsy DL2 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches)

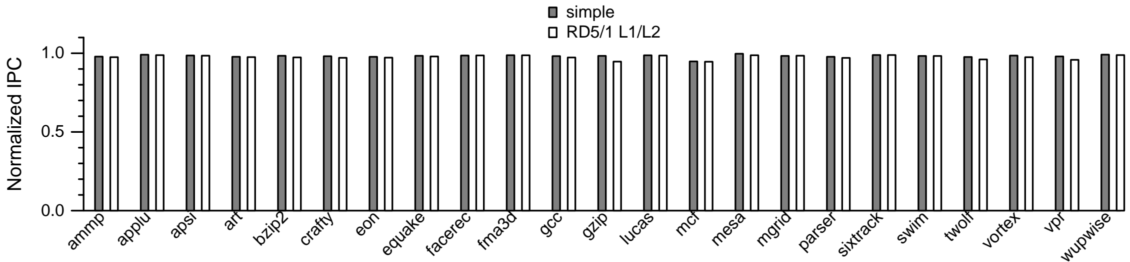

Figure 5.11: Drowsy IPCs for 1MB L2 (Normalized to Non-Drowsy Caches)

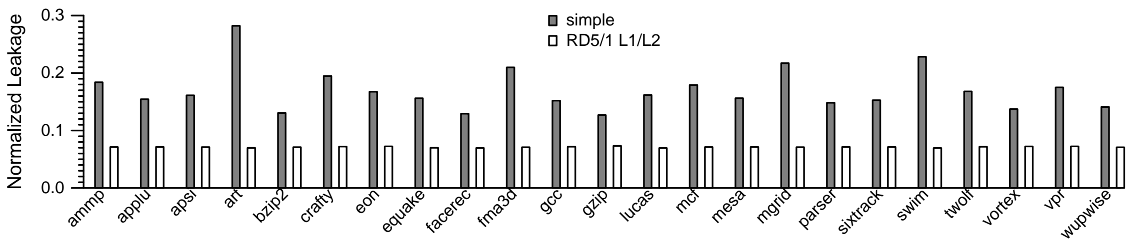

Figure 5.12: Drowsy DL1 Leakage for 1MB L2 (Normalized to Non-Drowsy Caches)

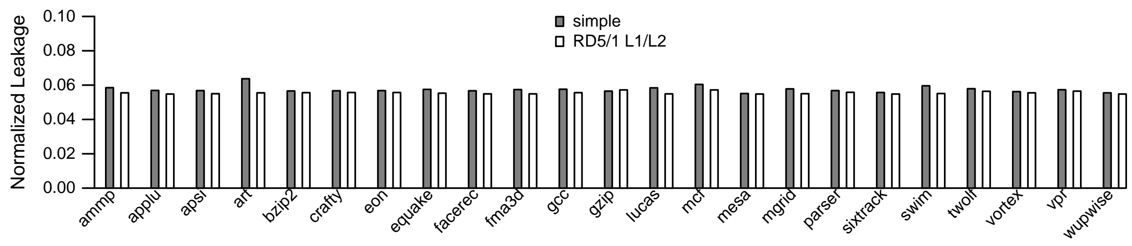

Figure 5.13: Drowsy DL2 Leakage for 1MB L2 (Normalized to Non-Drowsy Caches)

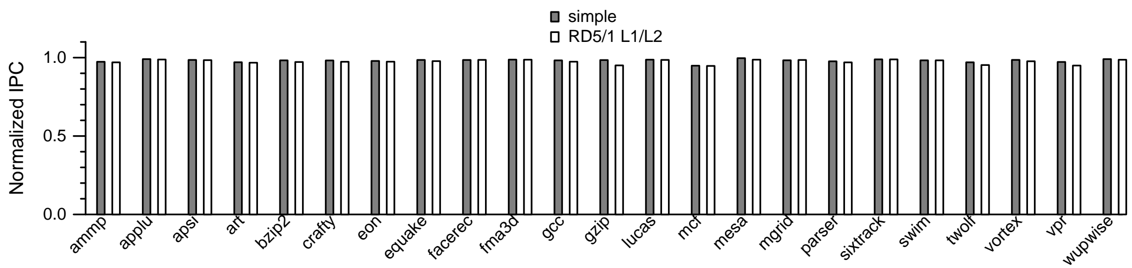

Figure 5.14: Drowsy IPCs for 2MB L2 (Normalized to Non-Drowsy Caches)

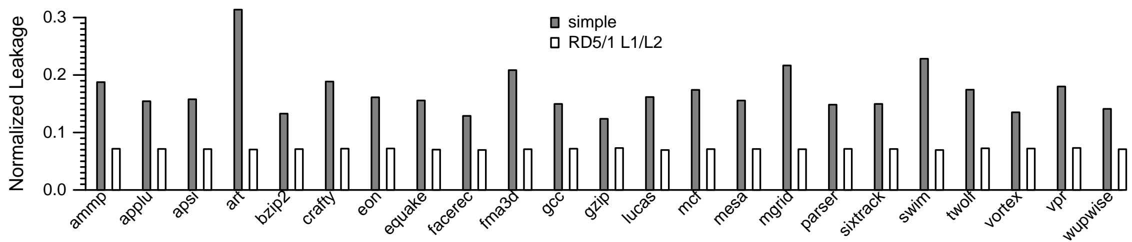

Figure 5.15: Drowsy DL1 Leakage for 2MB L2 (Normalized to Non-Drowsy Caches)

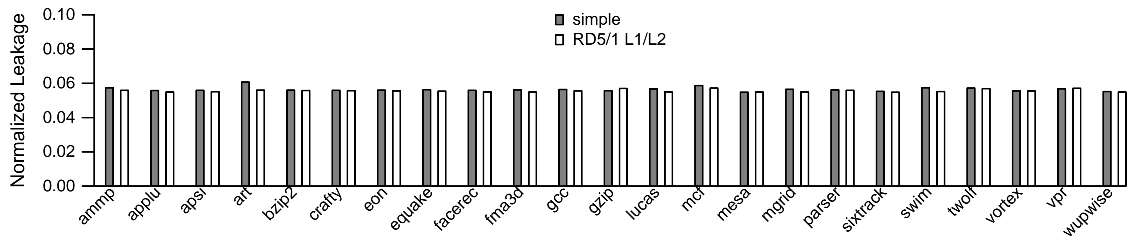

Figure 5.16: Drowsy DL2 Leakage for 2MB L2 (Normalized to Non-Drowsy Caches)

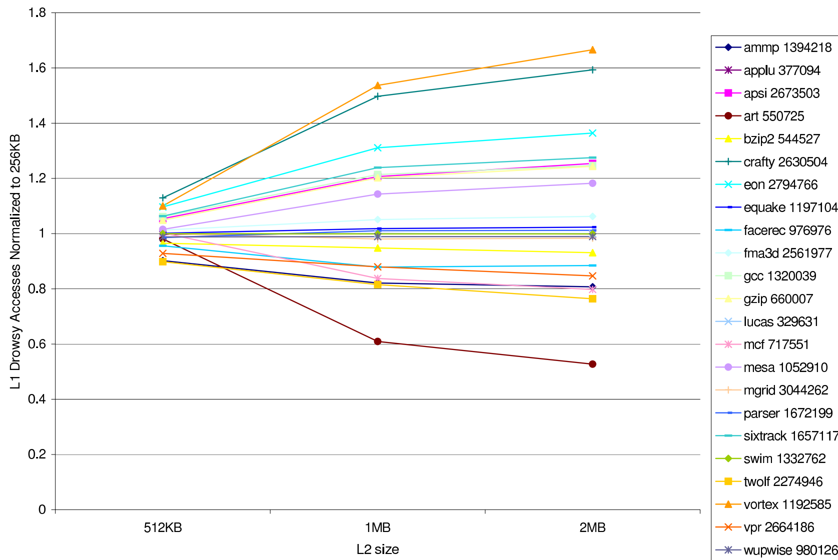

Figure 5.17: Drowsy Accesses For Simple Policy as L2 Scales

dependent. Unfortunately, the simple policy execution window parameter relies on the number of clock cycles, and thus the memory behavior for the L1 cache changes due to different L2 latencies for different cache sizes. Figure 5.17 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 size increases the number of drowsy accesses by 66% for vortex.

RD performance does not change with scaling, as illustrated by Figure 5.18: number of drowsy accesses across all benchmarks change 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 to 2MB.

The increases in L2 cache size result in increases in the number of drowsy accesses, since the 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

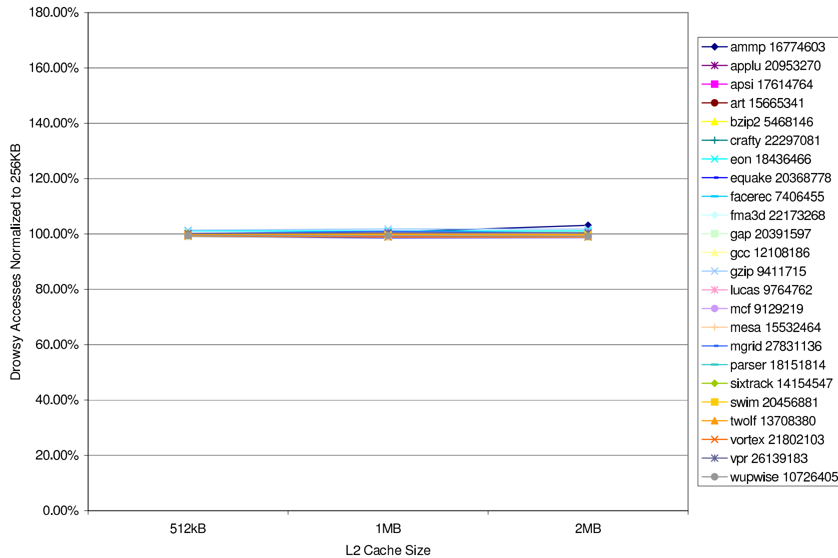

Figure 5.18: Drowsy Accesses For RD5 Policy as L2 Scales

and a net reduction in leakage (since the application finishes faster). The increased cache size does not result in significant increases in leakage, though, since the majority of the cache lines are already drowsy.

All figures previously analyzed indicate that the RD policy always results in lower power, but also delivers lower performance than the simple policy. However, the RD policy can be fine-tuned to provide lower power than simple and in some benchmarks even better performance. By having an RD of 50 for the L1 and L2, the performance improves since fewer lines are turned off. Benchmarks such as ammp in Figure 5.19 show better performance and lower power consumption in Figures 5.20 5.21 with the RD policy than simple. Further performance could be gained by reducing the L2 to having only one line active since it remains dormant most of the time, with most activity occurring in the L1.

Figure 5.22 shows the improvement in leakage of a 1.4GHz processor compared to a processor running at 500MHz with non-drowsy caches. Since HotLeakage does not model frequency, the leakage at the 1.4GHz processor has been scaled to accurately model the leakage in comparison to the 500MHz processor. Performance improves,

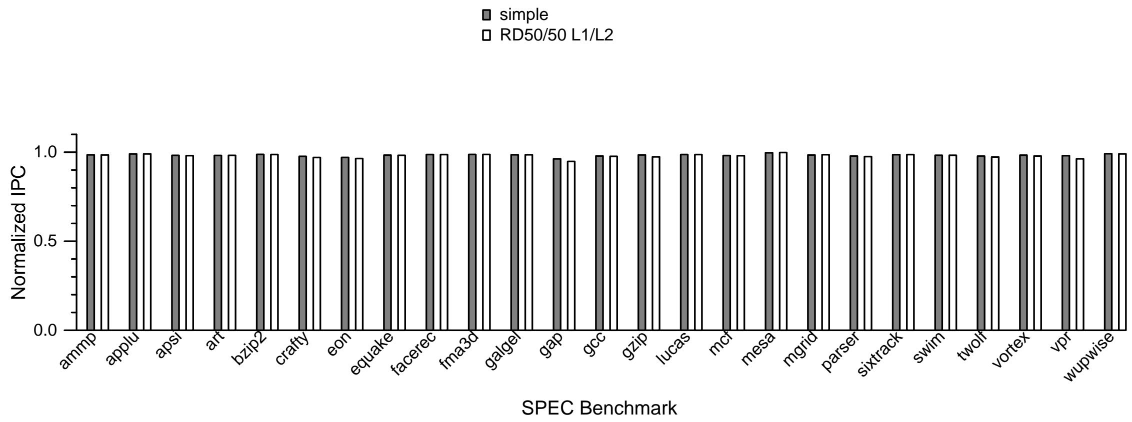

Figure 5.19: Drowsy IPCs for 512KB L2 (Normalized to Non-Drowsy Caches)

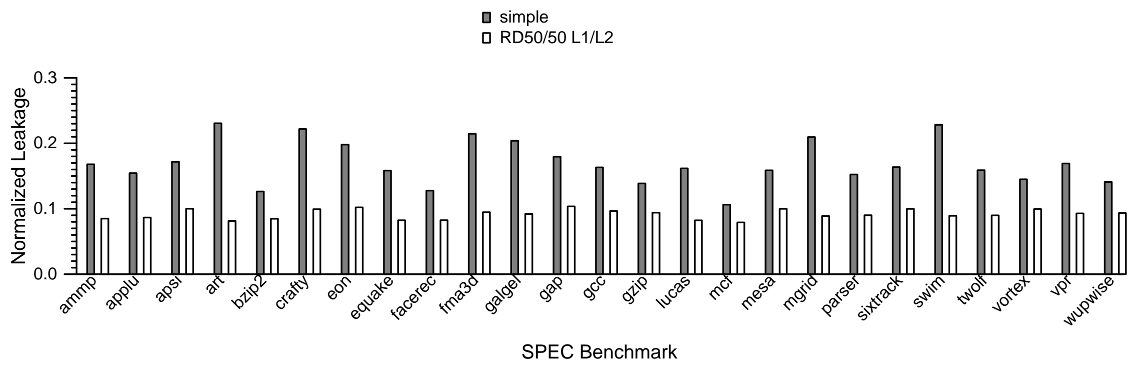

Figure 5.20: Drowsy DL1 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches)

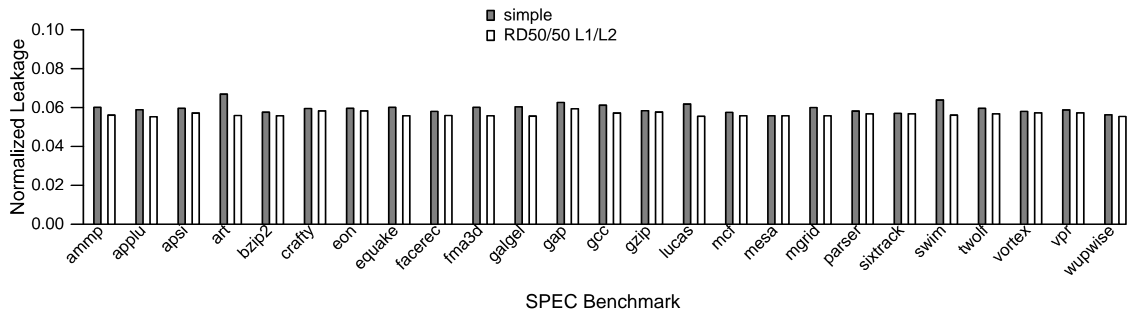

Figure 5.21: Drowsy DL2 Leakage for 512KB L2 (Normalized to Non-Drowsy Caches)

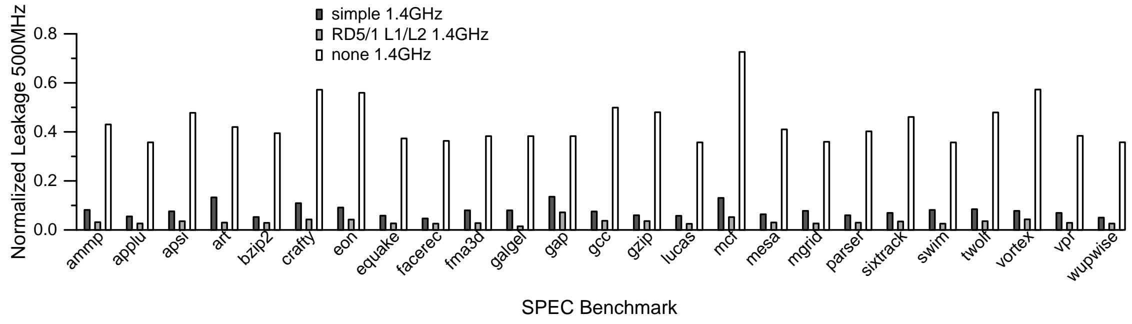

Figure 5.22: DL1 Leakage for 2MB L2 Normalized to 500MHz Non-Drowsy Caches

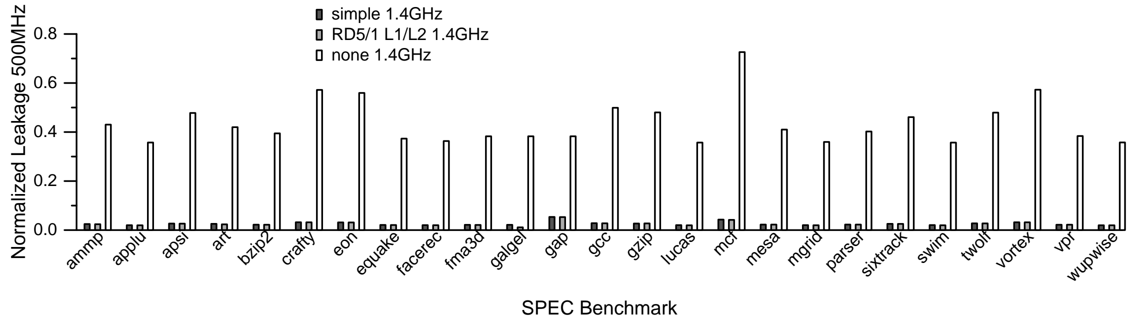

Figure 5.23: DL2 Leakage for 2MB L2 Normalized to 500MHz Non-Drowsy Caches

while power remains the same, and the net energy consumed is reduced. By accurately modeling the 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. The change in behavior for the simple policy can be attributed to the increased latency of the caches in raw clock cycles.

5.2 Embedded Architectures

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, no-access, 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 no-access 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.

Update window size for the other drowsy policies is 512 cycles for the heap and stack cache, and 256 cycles for the global cache[1]. 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. [8] and Petit et al. [24], suffer 20% more leakage energy.

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 5.24 shows the percentages of accesses that hit on the first lookup, hit on the 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 of both ways, the prediction bit is set to the way holding the LRU

[1] These values were found to be optimal for simple drowsy policies and region caches [10].

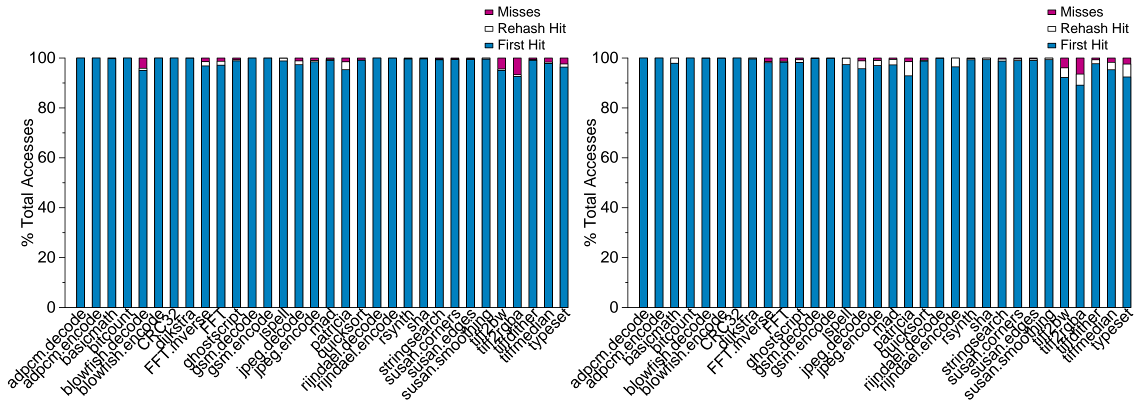

Figure 5.24: Heap Accesses Broken Down by Category (CA Cache on Left, MRU Cache on Right)

line to be evicted. On an incorrect guess or miss, MRU caches suffer an additional cycle of latency over a normal two-way associative cache. This is offset by significant power savings on a correct prediction: since the MRU cache is physically partitioned into two sequential sets, it only charges half the bitline length (capacitance) of a direct mapped or CA cache of the same size. Figure 5.24 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.

5.2.1 Sustaining Performance

Figure 5.25 graphs IPCs relative to direct mapped caches with the best update windows from Geiger et al. [10]. 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 config-

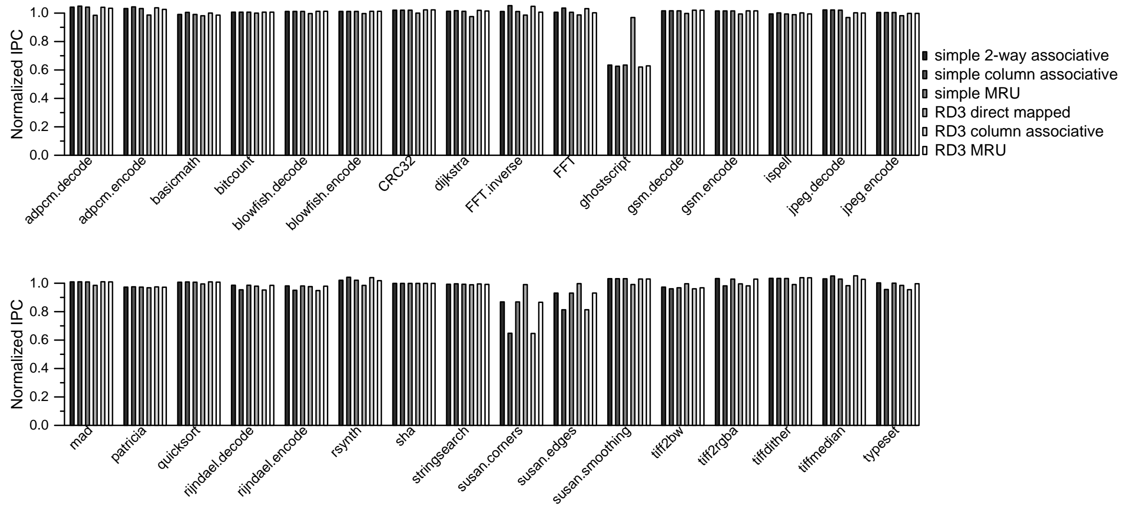

Figure 5.25: IPCs Normalized to *simple* Drowsy Direct Mapped Region Caches

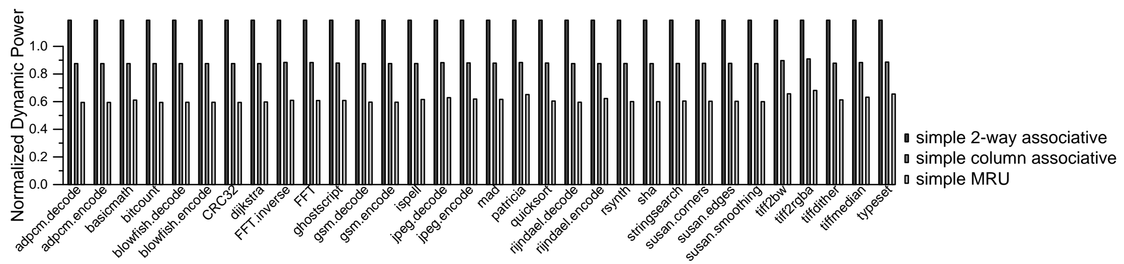

Figure 5.26: Dynamic Power Consumption of Associative Heap Caches Normalized to a Direct Mapped Heap Cache

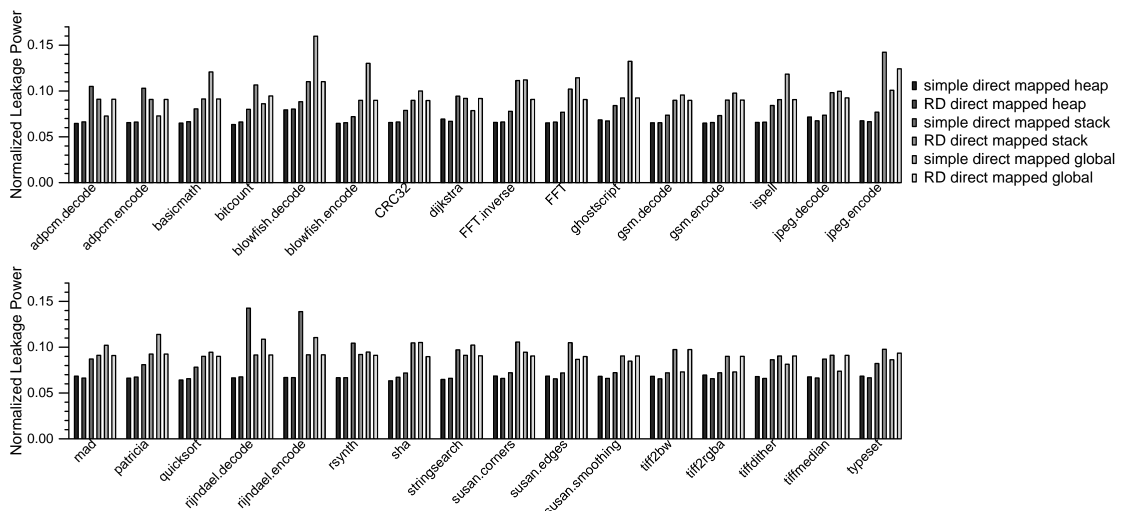

Figure 5.27: Static Power Consumption of Different Drowsy Policies with Direct Mapped Region Caches Normalized to Non-Drowsy Direct Mapped Region Caches

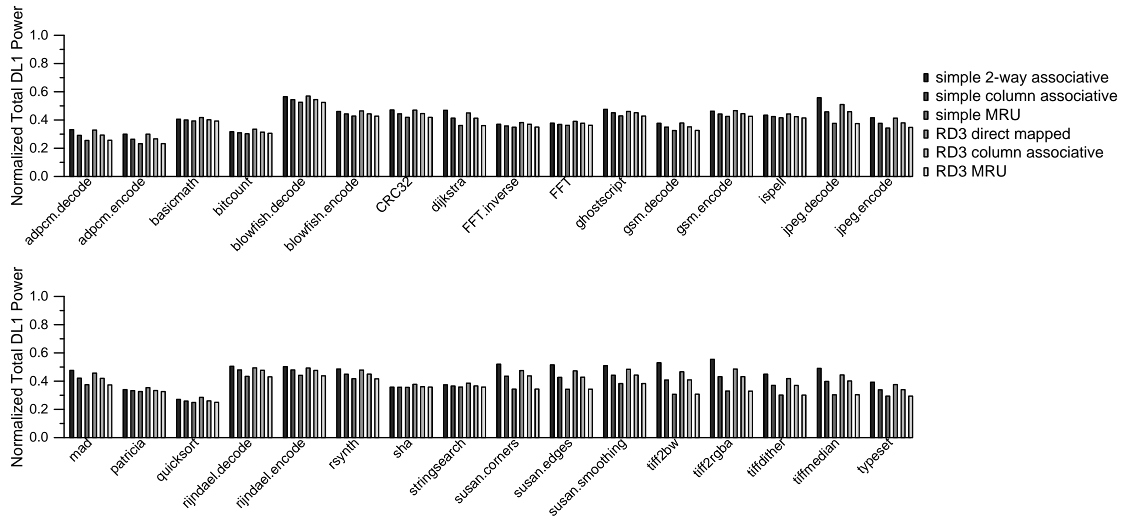

Figure 5.28: Total Power of Drowsy Policies (for All Region Caches) and Heap Cache Organizations Normalized to Direct Mapped Region Caches

uration, but a lower IPC than the direct-mapped cache configuration. This behavior is platform specific. We investigated this anomalous behavior and found pathological accesses patterns where the cache lines that are evicted have not been fetched yet from the L2. Thereby, new requests stall waiting for pre-existing requests to complete, resulting in a large backlog of requests. This hazard occurs more often with the associative cache than with the direct-mapped cache. Increasing associativity or increasing number of sets alleviates this. Porting the benchmarks 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 those benchmarks. Overall, IPCs are within 1% of the best baseline case, as with the simple and noaccess results of Flautner et al. [8]. These differences are small enough to be attributed to modeling error.

5.2.2 Reducing Dynamic Power

Figure 5.26 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 5.26 illustrates this for CA and MRU organizations: rehash checks and way mispredictions increase power consumption for some scenarios.

5.2.3 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 5.27. 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 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. Update window size for simple drowsy policies and RD buffer sizes are 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 5.26 represent a small contribution to the total L1 Dcache power consumption shown in Figure 5.28. 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 negligible performance reduction of 1.2%.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

We adapt techniques developed to improve cache performance, using them to address both dynamic switching and leakage power.

We have introduced a drowsy mechanism that is impervious to changes in memory latency and provides an upper and lower bound on expected power consumption. When this reuse distance (RD) policy is applied to the L1, drowsy prediction rate is retained 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 to the simple policy, while suffering 56% less leakage power. 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.

With respect to drowsy caching for embedded applications, 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 up-date 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.

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. 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.

BIBLIOGRAPHY

[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, June 2000.

[4] D. Burger and T. Austin. The simplescalar toolset, version 2.0. Technical Report 1342, University of Wisconsin, June 1997.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] H. Lee. *Improving Energy and Performance of Data Cache Architectures by Exploiting Memory Reference Characteristics*. PhD thesis, University of Michigan, 2001.

[21] 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.

[22] 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.

[23] 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, May 2005.

[24] 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.

[25] 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.

[26] 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.

[27] A. Seznec. A case for two-way skewed-associative cache. In *Proc. 20th IEEE/ACM International Symposium on Computer Architecture*, page 169, May 1993.

[28] 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.

[29] 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.

[30] R. Sites. It's the Memory, Stupid! *Microprocessor Report*, 10(10):2–3, Aug. 1006.

[31] A. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, Sept. 1982.

[32] C. Su and A. Despain. Cache designs for energy efficiency. In *Proc. 28th Annual Hawaii International Conference on System Sciences*, pages 306–315, Jan. 1995.

[33] 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.

[34] 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.

[35] 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.