Drowsy Region-Based Caches: Minimizing Both Dynamic and Static Power Dissipation

Michael J. Geiger
Advanced Computer Architecture Lab
The University of Michigan
Ann Arbor, MI 48109-2122
geigerm@eecs.umich.edu

Sally A. McKee
Computer Systems Lab
Cornell University
Ithaca, NY 14853-3801
sam@csl.cornell.edu

Gary S. Tyson
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
tyson@cs.fsu.edu

ABSTRACT
Power consumption within the memory hierarchy grows in importance as on-chip data caches occupy increasingly greater die area. Among dynamic power conservation schemes, horizontal partitioning reduces average power per data access by employing multiple smaller structures or using cache subbanks. For instance, *region-based caching* places small caches dedicated to stack and global accesses next to the L1 data cache. With respect to static power dissipation, leakage power may be addressed at both circuit and architectural levels. *Drowsy caches* reduce leakage power by keeping inactive lines in a low-power mode. Here we merge drowsy and region-based caching to reduce overall cache power consumption, showing that the combination yields more benefits than either alone. Applications from the MiBench suite exhibit power reductions in the cache system of up to 68-71%, depending on memory configuration, with a small increase in execution time.

Categories and Subject Descriptors
B.3.2 [**Memory Structures**]: Design Styles—*cache memories*.

General Terms
Design, Performance.

Keywords
Region-based caches, drowsy caches, energy-aware design.

1. INTRODUCTION
Reducing energy consumption remains a major design goal for architects of embedded systems and mobile computing devices. The memory subsystem comprises a large part of the die area of these increasingly prevalent systems, and caches may consume over 40% of a chip's overall power [12]. Minimizing total power requires reducing both dynamic and static power consumption, particularly in the on-chip memory hierarchy.

Reducing dynamic energy has largely focused on partitioning caches into small, low-power structures [4][9][15]. At the architectural level, *region-based caching* exploits locality within stack and global data accesses by directing them to separate caches [10]. The smaller, dedicated structures reduce average power per data access, and do so without increasing execution time, since high reference locality in the stack and global regions yield high hit rates.

As feature sizes shrink, leakage power constitutes an increasing fraction of total processor power. This affects cache structures disproportionately, since they have large area requirements (that tends to increase leakage effects) and most cache lines are only infrequently accessed (minimizing dynamic energy requirements) [6][14][17]. Projections show that leakage may constitute as much as 70% of total consumed cache power in a 0.07 micron process [7]. One promising technique for reducing leakage is *drowsy caching* [3], a memory design that exploits dynamic voltage scaling: reduced supply voltage to inactive lines lowers their static power dissipation. When such a drowsy line is accessed, the supply voltage must first be returned to its original value. Lines may change state from active to drowsy after an interval of a number of cycles; depending on the policy, lines accessed within the interval may remain active. Drowsy caches save less power than many other leakage reduction techniques, but do not suffer the dramatically increased latencies of other methods.

Our region-based caching scheme places data that is not in the stack or global regions into the L1 data cache. Most references to this structure are heap accesses, thus we refer to it as the *heap cache*. Largest of the three structures, it only averages 30% of all requests. This cache always contains many inactive lines, and thus dissipates much static power. All three region-based structures will likely benefit from leakage-reduction techniques (Figure 3 and Figure 7 support this).

In this paper, we investigate applying drowsy caching techniques to reduce static power in region-based caching systems. These techniques are complementary, not only because they target different aspects of the power problem, but because each approach improves the performance of the other. Drowsy caching can reduce the static power increase caused by additional region caches, while the partitioning strategy employed by region caching allows more aggressive drowsy intervals and policies to be employed. Section 2 discusses prior research on power-saving techniques for caches, including a detailed overview of region caching and drowsy caches. Section 3 describes the experimental setup we use to evaluate the effectiveness of combining these approaches, and highlights our results. Section 4 presents conclusions and future directions for this research.

CF '05, May 4–6, 2005, Ischia, Italy.

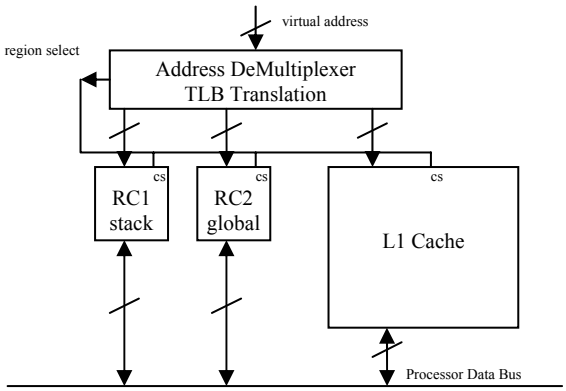


Figure 1: Memory design for region-based caching [11]

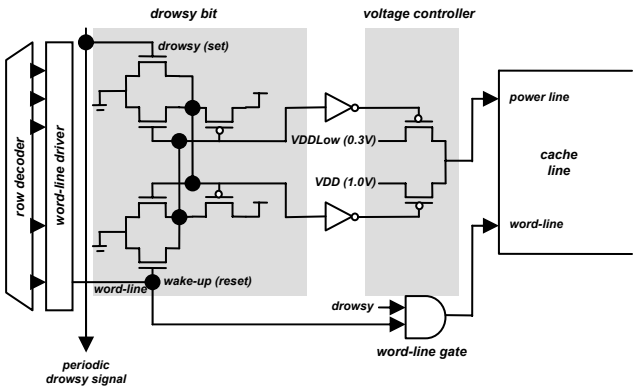


Figure 2: Drowsy cache line [8]

2. BACKGROUND

Cache partitioning schemes generally fall in two categories: vertical and horizontal. Since dissipated power per access is proportional to cache size, partitioning techniques reduce power by directing accesses to smaller structures. Vertical partitioning adds a level closer to the processor than the L1, as in line buffers [4][15] and filter caches [9]. These provide low-power access for data with temporal locality, but typically suffer many misses requiring L1 accesses, thus they can increase average L1 latency. Horizontal partitioning splits entities (e.g., lines) into smaller segments, as in cache sub-banking [4][15]. References are routed to the proper segment, reducing the power requirements for data access (dynamic power).

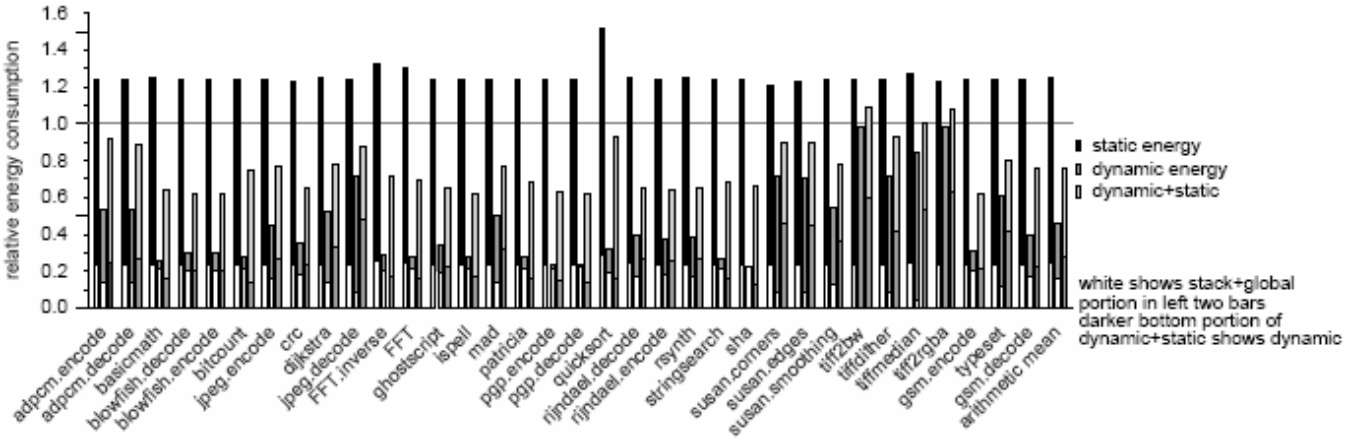
Region-based caching [10] exploits the high locality of stack and global data to horizontally partition caches at a larger granularity. Illustrated in Figure 1, small caches are added at the level of the L1 data cache. Stack and global region accesses are directed to the appropriate caches; all other data accesses go to the L1, as usual. Since most remaining accesses are to heap data, with a small number of accesses to text and read-only regions, we refer to the L1 as the *heap cache* in region-based caching. On a memory reference, only the appropriate region cache is activated and draws power. Relatively small working sets for stack and global regions allow their caches to be small, dissipating even less power on hits. Since about 70% of references are in these regions[1], dynamic power consumption drops significantly when comparing region caches to a unified L1 data cache. Splitting references among caches eliminates inter-region conflicts, thus each cache may implement lower associativity, reducing complexity and access time.

The relative cost of leakage power increases with shrinking feature size and new circuit technologies. Approaches to reduce leakage using gated-V_{DD} techniques [6][14][17] selectively turn off cache lines with data unlikely to be reused. Once turned off, line state is lost, and the cost of reloading from L2 cache may negate leakage power savings. Furthermore, added latency for accessing L2 may adversely impact performance. To maintain acceptable performance, this policy is implemented conservatively with complex algorithms that minimize costs.

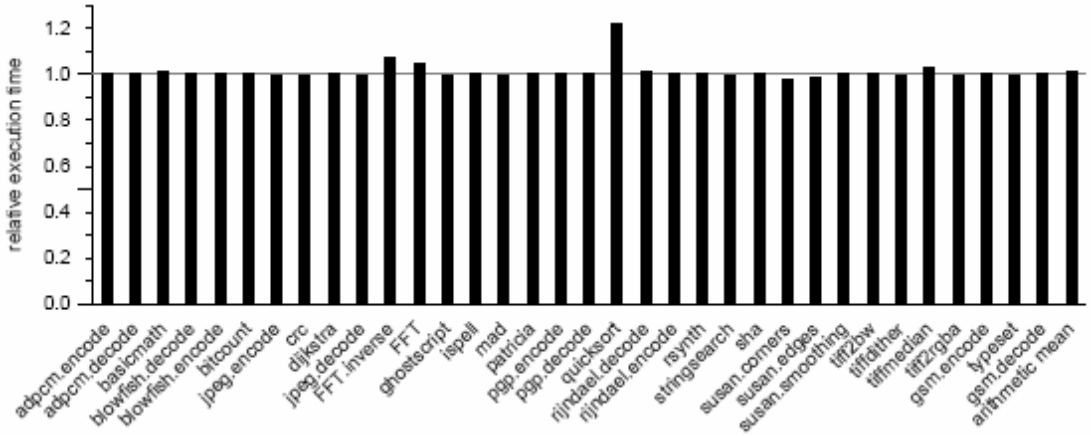
By contrast, drowsy caching is a state-preserving technique; rather than turning off inactive lines, a drowsy cache places them into a low-power mode. When a drowsy line is referenced, that line merely needs to return to full power before satisfying the access. Although this method does not save as much leakage power as a gated-V_{DD} approach, it avoids performance penalties of extra L2 accesses. Drowsy caching can thus be more aggressive at moving lines into a low-power state, increasing potential savings. Multiple circuit techniques could implement drowsy caching, but we assume dynamic voltage scaling as proposed in Flautner et al. [3]. Figure 2 shows the hardware to implement a drowsy cache line—a drowsy bit, a voltage control mechanism, and a wordline gating circuit. The voltage controller switches the line's supply voltage between high (active) and low (drowsy) values depending on the drowsy bit state. The wordline gate prevents accesses while in drowsy mode, avoiding destruction of a drowsy line's data. When accessed, a drowsy line's bit is cleared, "awakening it" and returning the supply voltage to its active (high) value. If tags are kept drowsy, they may need to be awakened, possibly increasing wakeup latency. Direct-mapped caches derive no benefit from keeping tags awake since there is only one line per index. We model direct-mapped caches and assume drowsy tags.

Flautner et al [3] present two policies for setting the per-line drowsy bits. In the *simple* policy, all lines become drowsy after a certain update interval. In the *noaccess* policy, only lines not accessed within the interval become drowsy. The *simple* policy reduces leakage power more effectively than the *noaccess* policy, since the former moves lines from active to drowsy more aggressively. However, it is locally oblivious, and may increase execution time when lines soon to be accessed are placed into drowsy mode. Flautner et al. [3] find a minimal difference in performance among policies.

[1] Our 70% figure agrees with Lee and Tyson [10], but differences in application suites yield a different distribution among stack and global references.



(a)



(b)

Figure 3: Energy consumption (a) and performance (b) of region-based caches compared to single 32KB DM L1 cache

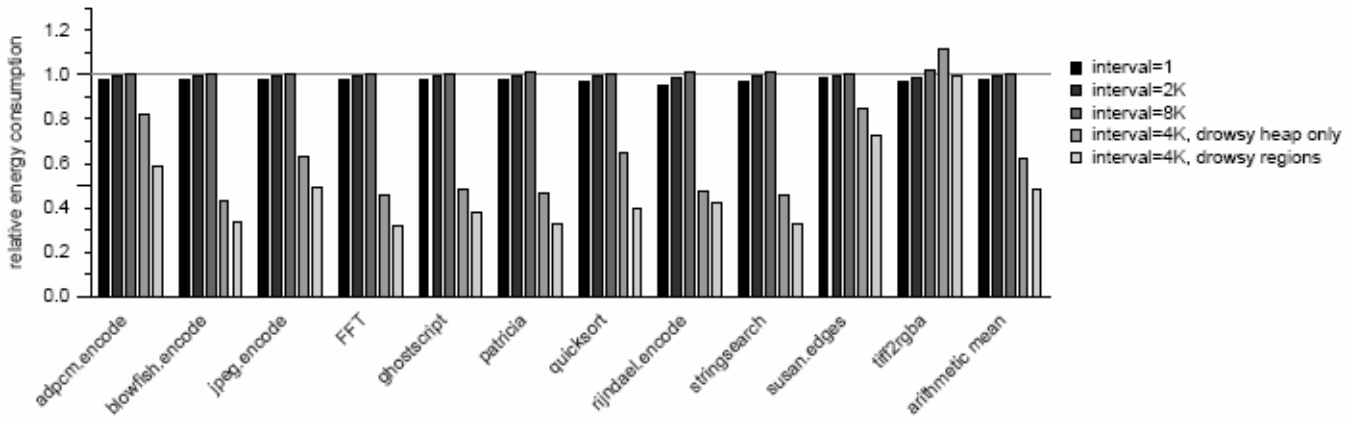
3. EXPERIMENTS

We use a modified version of the SimpleScalar ARM model [1]. Dynamic power is modeled using Wattch [2]. Static power is calculated using Zhang et al.'s HotLeakage [16], which contains a detailed drowsy cache model used by Parikh et al. to compare state preserving and non-state-preserving techniques for leakage control [13]. HotLeakage tracks the number of lines in both active and drowsy modes and calculates leakage power appropriately. It models the dynamic and leakage power of additional required hardware: a global counter to track the update interval, local counters for the *noaccess* policy, and circuitry per line to support a drowsy state. In all cases, the processor configuration is an in-order model similar to the Intel StrongARM SA-110 [12]. All our drowsy caches implement the *simple* policy.

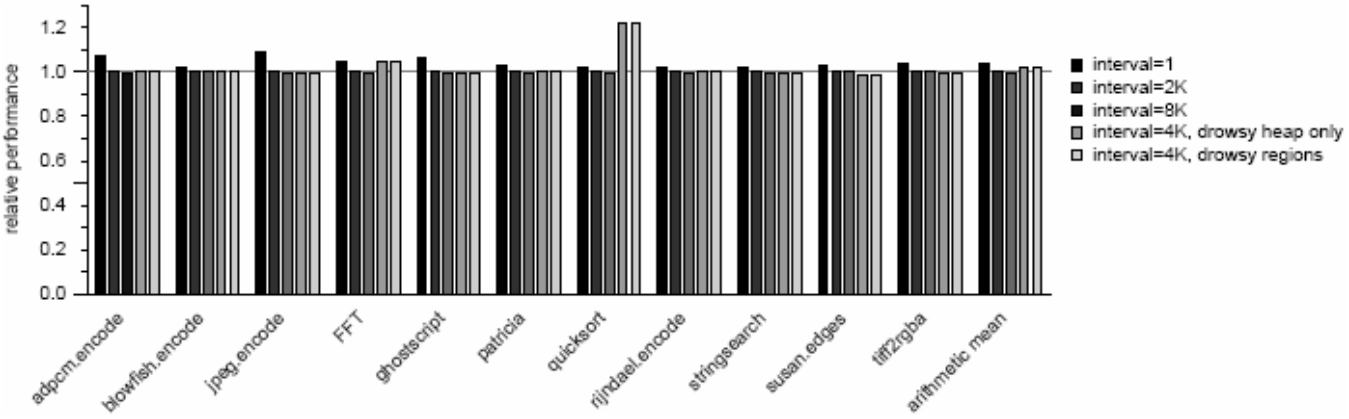
3.1 Region-Based Caching

Figure 3(a) and (b) show simulation results of region-based caching for applications from the MiBench suite[5]. Average values in all graphs (including those showing a subset of the applications) represent arithmetic means across the entire suite.

Our region caching system uses a 4 KB L1 stack cache, a 4 KB L1 global cache, and a 32 KB L1 data cache; all three are direct-mapped. We compare this configuration against a baseline direct-mapped 32 KB unified cache (see Lee and Tyson for comparisons with other configurations [10]). Figure 3(a) shows relative power consumption for the region caches. Three vertical bars are presented for each application: they indicate the change in leakage energy, switching (dynamic) energy, and total cache energy. Each of these numbers is normalized to the corresponding value for the baseline; for example, the static power bars show the ratio of static energy consumption in our region-based caches to the static energy consumption in the baseline. Within the left two bars, we indicate energy consumed by the stack and global caches (in white) and the heap cache (shaded). We show the ratio between static and dynamic energy contributing to the total cache energy in the third bar. The first bar shows that leakage energy increases in our region-based caching (due to the two extra caches, which together are one fourth the size of the baseline cache) by 25.2% on average. However, this increase is overwhelmed by a dramatic savings in dynamic energy, resulting in a 23.6% total power savings compared to the unified baseline cache.



(a)



(b)

Figure 4: Energy consumption (a) and performance (b) for varying drowsy intervals and drowsy region caches compared to a 32KB drowsy L1 with 4K-cycle interval

Overall savings is less than Lee and Tyson find in the original region cache study [10] because our ratios of dynamic to static power differ. Our application and pipeline configuration choices significantly increase leakage power's impact on total power. The dramatic relative increase in leakage power in our applications can be attributed to two factors. First, only 26% of all instructions in MiBench reference memory, versus 56% in the SPEC CPU2000 suite. Second, programs running on an in-order core have inherently longer delays between memory accesses than they do on an out-of-order machine. Activity in the L1 data cache is therefore extremely low when compared to the original study, reducing the relative contribution of dynamic power to total power. Infrequently active caches benefit greatly from the leakage reduction drowsy caching provides. Adding drowsiness to region caching should amplify the power savings by reducing static power consumption.

Figure 3(b) shows the relative performance impact of region caching on these applications. In most cases, we see small speedups. The slightly larger capacity of our region caching configuration is one reason for the speedups. In addition, since region-based caching reduces conflicts, average memory latency goes down. Nonetheless, we have not tuned the region cache sizes for MiBench applications, and other configurations may yield larger speedups. For the current configuration, the impact of an individual application, *quicksort* (whose execution slowdown is 22.5% due to a large increase in global cache misses), yields an average slowdown of 1.1% for region caches vs. the 32KB unified baseline cache.

3.2 Drowsy Region-Based Caching

Figure 4(a) and (b) show simulation results for drowsy caching and region-based caching for a representative subset of the MiBench suite. Here the baseline is again a 32KB direct-mapped unified cache, but made drowsy with a 4K-cycle update interval (i.e., after every 4K cycles, all lines are put into drowsy mode). In Figure 4(a), the left three bars for each application indicate relative total energy for 32KB unified caches with update intervals of one cycle (always drowsy), 2K cycles, and 8K cycles. The right two bars show relative total energy for drowsy region caches with a 4K cycle update interval. For the first of these, only the heap cache is drowsy, whereas for the second, all three region caches are drowsy. Changing the update interval has little impact on total energy consumption of the unified 32KB direct-mapped cache (leakage energy increases with increasing intervals, as ex-

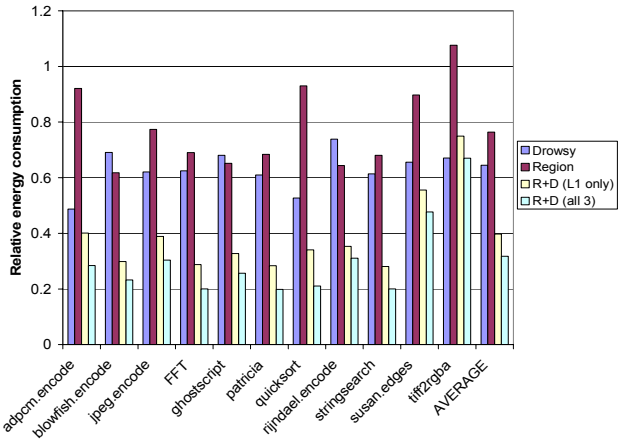


Figure 5a: Energy consumption of combined region and drowsy caching

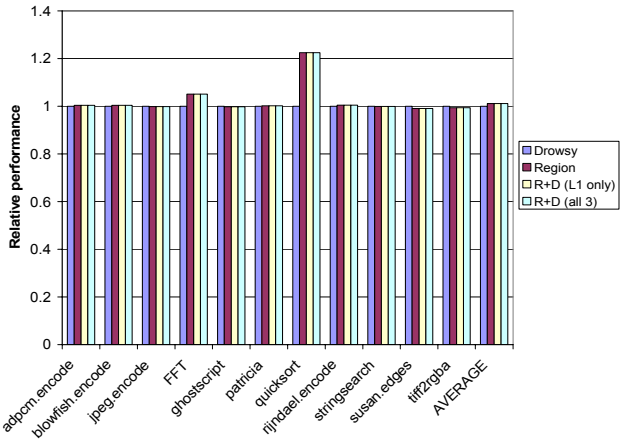


Figure 5b: Performance of combined region and drowsy caching

pected, but dynamic energy remains almost constant, and now dominates total energy consumption). In contrast, going from a drowsy unified cache to a drowsy region cache configuration yields much bigger energy savings. Making just the heap cache drowsy yields a 38.1% energy savings, whereas making all regions drowsy brings total savings up to 50.9%. Changing update intervals for the region caches has minimal effect on total energy.

Figure 4(b) shows the performance impact of changing update intervals or adding region caches. Recall that the baseline is a 32KB direct-mapped unified drowsy cache with a 4K-cycle update interval. One visible trend is that performance improves for larger update intervals, at the expense of slightly higher total power consumption. If region-based caching is not used for these applications, any of the larger intervals represents a reasonable design choice. Drowsy region caching reduces performance compared to the unified baseline cache with the same update interval, but only by 1.1% on average.

Figure 5(a) and (b) compare energy and performance of a baseline 32KB direct-mapped unified L1 cache to a drowsy version with a 4K-cycle update interval and to our region cache configuration (4KB stack cache, 4KB global cache, and 32KB heap cache) under three different drowsiness schemes. First we look at region caching with no drowsiness, then with only the heap cache drowsy, then with all three regions drowsy. Figure 5(a) shows total cache energy compared to the non-drowsy, unified baseline. The combination of region and drowsy caching significantly reduces overall energy consumption. A drowsy heap cache with standard stack and global region caches reduces total energy by 60%, and making all region caches drowsy reduces energy by 68%. Drowsy caching alone reduces total energy by only 35%, and region caching alone by only 24%.

Combining techniques increases leakage energy over drowsy caching alone, especially when the stack and global caches are not drowsy. When all regions are drowsy, the increased cache capacity leads to a small increase of about 0.4% in leakage energy: incorporating drowsy support to all caches almost totally eliminates the increase in leakage power due to the greater number of cache lines in region-based caching.

Figure 5(b) shows that the performance impact of combining techniques is very small. Since drowsy caching has a negligible impact on performance, the runtime penalty is effectively equal to the cost of region caching—about 1%. Performance drops in only two applications (*FFT* and *quicksort*), and in each case the cause is a dramatic increase in global cache misses.

As noted above, varying the update interval has little effect on total power for both unified and region caches. However, the effect on performance is more pronounced, as shown in Figure 4(b). To find the best interval for each cache, we vary the interval from 1 to 8000 cycles and plot the resulting power/performance curve. For a unified L1 cache, 512 cycles works best. For region caches, the ideal interval differs for each region—512 cycles for the stack and heap, 256 cycles for the global region. The fact that the heap cache has the same ideal interval as the stack cache is somewhat surprising; we expect data with low locality to have a low ideal interval. This result suggests that some heap data possesses a similar degree of locality to stack data. We leverage this fact in Section 3.3 to show that we can move hot heap data into its own cache, allowing us to aggressively apply drowsy caching to the remainder of the heap.

Region and drowsy caching function well together, and our figures demonstrate this, but they do not highlight the techniques' ability to perform better together than individually. When comparing a 32KB unified direct-mapped drowsy cache with a 512-cycle update interval to region caches configured with a 512-cycle update interval for the heap and stack caches and 256-cycle interval for the global cache, our experimental results show that the reduction in energy from the combined techniques is greater than the sum of reductions for each technique alone—68% versus 59% (35%+24%). To understand this, we examine how region caching achieves its 24% energy reduction. Region caches are much more effective at reducing dynamic energy, achieving a 64% reduction in dynamic energy, but this reduction is offset by an increase of 25% in static energy. Total energy saving is thus smaller—24% on average. However, drowsy caches effectively eliminate most static energy dissipation, so the dynamic energy savings of region caches is no longer offset by increases in static energy. In fact, when we count the number of cache lines not in drowsy mode (averaged over each cycle of execution), the unified drowsy cache averages 9.5 non-drowsy lines versus only 8.2 for drowsy region caches, even though the latter organization increases the effectiveness of the drowsy selection hardware. This

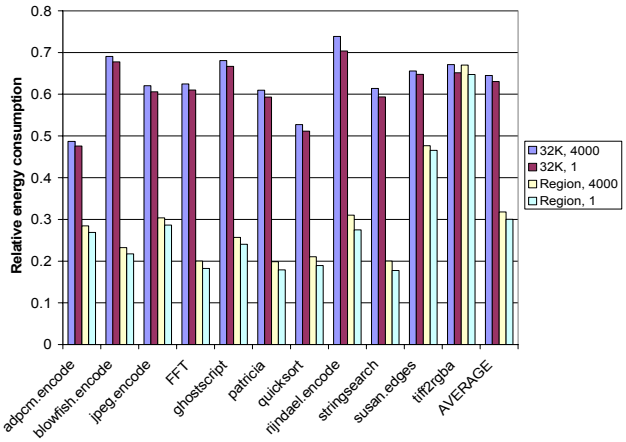


Figure 6: Comparison of large and small windows

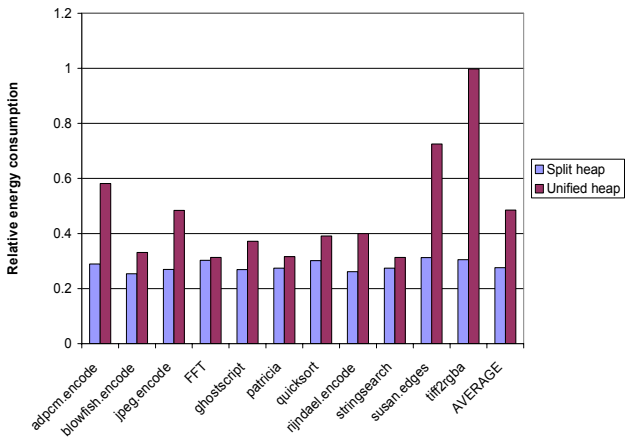


Figure 7: Energy consumption of split and unified heap schemes vs. single drowsy L1 data cache

means that the static power that would be increased by the additional caches is effectively eliminated, enabling the full benefit of the dynamic power savings to be realized.

3.3 Varying Configurations

Splitting the caches by region groups together data with similar locality. Stack data tends to display very high locality, and active lines will stay active for a period of time. Once those lines become inactive, however, they tend to remain so for a significant period. At that point, we can safely place them into drowsy mode without their being accessed in the near future. By contrast, most lines in the heap region and other regions that reside in the L1 data cache exhibit low locality. Multiple accesses to a line in a short interval are rare, meaning that once a line is accessed, it is unlikely to be accessed again for many cycles. Capitalizing on these traits advocates moving heap cache lines into drowsy mode soon after being accessed.

Region-based caching already allows us to be more aggressive with drowsy caching policies because it splits the reference stream and ensures that each cache only sees a portion of the total data accesses. If we further consider the locality characteristics of each region, we see that we can be very aggressive with the lines in the heap cache, which tend to be referenced rarely, and still relatively aggressive with the lines in the stack cache once we know they have become inactive. This argues for experimenting with the *noaccess* policy for determining when lines become drowsy.

Increasing the aggressiveness of our drowsy caching policy means decreasing the update interval, and this change has a positive effect on the circuit overhead of the drowsy cache as well. A shorter interval implies a smaller cycle counter. If the interval shrinks to one—meaning that a line is put into drowsy mode directly after being accessed—the structure of the drowsy cache line also changes. The drowsy bit is no longer necessary, since each line is either active or drowsy. It is true that such an aggressive policy will increase the number of accesses that must pay the penalty for switching a cache line from drowsy to active. However, data with high locality should not suffer as much as might be expected, as repeated accesses to a line would keep that line active and reduce the number of mode switches.

Figure 6 highlights the difference in energy between typical drowsy caches with a large update interval and aggressive, always drowsy caches. We compare a 4K-cycle update interval to a 1-cycle interval for both a single L1 data cache and for our typical region-based caching configuration, using a non-drowsy 32KB direct-mapped cache as the baseline. This graph primarily shows the added benefit of combining drowsy and region caches. When the cache configurations are identical, total power consumption is similar across interval sizes. Reduced interval size has a significant impact on leakage power. For a 1-cycle interval, leakage energy is 29% less than it is with a 4K-cycle window, leading to a 1.6% decrease in total energy, as shown in the figure. This decrease in energy does not greatly affect performance: both configurations suffer a slowdown of roughly 1%.

Another means of increasing aggressiveness of our drowsy region caching is to add a fourth region. Our experimental data suggest that not all heap data have low locality. If we can isolate the hot heap data, we can place it in a smaller cache to minimize the dynamic power. This data likely possesses similar locality characteristics to stack data, so we can use a degree of aggressiveness similar to that of the stack cache when making the new cache drowsy. Low-locality heap and other data still resides in the larger heap/L1 data cache. Since this large structure will see even fewer accesses and will (ideally) not contain performance-critical data, we may keep it drowsy at all times to minimize static power consumption.

To identify hot heap data, we profile the applications and examine intervals between accesses. This metric primarily demonstrates temporal locality of individual lines, but does not account for spatial locality between lines. In particular, we look at the distribution of intervals, choosing lines for which most accesses occur a few cycles apart. For our simulations, we build a table to route heap data to the appropriate cache. In practice, however, we would use two different heap allocators to place data in the correct address ranges. Feedback-directed compiler tools thus bear the burden of profiling the application and choosing proper data placement.

Splitting the heap provides a substantial power savings over the best drowsy three-region cache configuration we observed, given our cache sizes and applications (using 512-, 256-, and 512-cycle

intervals for the stack, global, and heap regions, respectively, as in our performance analysis at the end of Section 3.2). We use the same drowsy intervals for the stack and global caches in both cases—512 and 256 cycles, respectively. Our baseline is a 32 KB direct-mapped L1 data cache with a 512-cycle drowsy interval. As Figure 7 shows, average energy consumption of the split-heap scheme is 28.3% of the baseline value—a 71.7% reduction. By comparison, our best drowsy region-based caching scheme reduces energy by an average of 52.5% better than a unified drowsy cache. Almost all of the additional power savings comes from the reduced dynamic power requirements of routing that part of the heap data that has high locality to a second, smaller heap region cache. The cost of this technique includes software as well as hardware, since it requires compiler support for determining heap reference locality and support for dynamic memory allocation on two different heaps.

4. CONCLUSIONS

We have shown that the combination of two techniques for reducing memory system power, region-based caching and drowsy caching, can have a benefit that is greater than the sum of their parts. Both methods achieve significant reductions in their targeted domains—dynamic power for region-based caches, leakage power for drowsy caches. Because region-based caching splits the reference stream into groups with similar locality, the activity of the separate caches is well defined. Stack cache lines are very active for a certain period of time because stack data has high locality; once they become inactive, however, the lines remain inactive for many cycles. At the other end of the spectrum, the L1 data cache holds mostly heap data, which lacks locality. This structure must be large enough to maintain a high hit rate for this data, meaning that most of its lines are inactive most of the time. Drowsy caching techniques exploit both types of inactivity. They aggressively move L1 data cache lines into drowsy mode, preventing them from constantly leaking energy at a high rate. With an appropriate update interval, a drowsy stack cache allows its lines to remain in active mode as long as they are accessed, and then puts them into drowsy mode once the period of high activity has ended. The result is a significant reduction in energy consumption—as high as 68%—with a minimal performance penalty. We also show that further partitioning of the heap can provide even greater benefits (up to 71% savings in total energy) when heap references with high locality can be serviced by a second, smaller heap cache. Although we target this paper toward embedded systems, we believe that merging these techniques can have a major benefit for high-performance computing as well.

5. REFERENCES

[1] T. Austin. SimpleScalar 4.0 Release Note. http://www.simplescalar.com/.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. *Proc. 27th Int'l Symp. on Comp. Arch.*, pp. 83-94, June 2000.

[3] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. *Proc. 29th Int'l Symp. on Comp. Arch.,* pp. 147-157, May 2002.

[4] K. Ghose and M. B. Kamble. Reducing Power in Superscalar Processor Caches using Subbanking, Multiple Line Buffers and Bit-Line Segmentation. *Proc. Int'l Symp. on Low Power Electronics and Design*, pp. 70-75, August 1999.

[5] M. R. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *IEEE 4th Workshop on Workload Characterization*, pp. 3-14, December 2001.

[6] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. *Proc. 28th Int'l Symp on Comp. Arch.*, pp. 240-251, June 2001.

[7] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy Instruction Caches: Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction. *35th Symp. on Microarch.*, pp. 219-230, November 2002.

[8] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power. *IEEE Transactions on VLSI*, Vol. 12, No. 2, pp. 167-184, February 2004.

[9] J. Kin, M. Gupta, and W. H. Mangione-Smith. Filtering Memory References to Increase Energy Efficiency. *IEEE Transactions on Computers*, Vol. 49, No. 1, pp. 1-15, January 2000.

[10] H-H. S. Lee and G. S. Tyson. Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors. *Proc. Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Sys.*, pp. 120-127, November 2000.

[11] H-H. S. Lee. Improving Energy and Performance of Data Cache Architectures by Exploiting Memory Reference Characteristics. Doctoral thesis, Univ. of Michigan, 2001.

[12] J. Montanaro, et al. A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. *Digital Technical Journal*, Vol. 9 No. 1, pp. 49-62, January 1997.

[13] D. Parikh, Y. Zhang, K. Sankaranarayanan, K. Skadron, and M. Stan. Comparison of State-Preserving vs. Non-State-Preserving Leakage Control in Caches. *Proc. 2nd Annual Workshop on Duplicating, Deconstructing, and Debunking*, pp. 14-24, June 2003.

[14] M. Powell, S-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. *Proc. Int'l Symp. on Low Power Electronics and Design*, 2000, pp. 90-95, July 2000.

[15] C-L. Su and A. M. Despain. Cache Designs for Energy Efficiency. *Proc. 28th Hawaii International Conf. on System Science*, pp. 306-315, January 1995.

[16] 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, March 2003.

[17] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode-control: A Static-Power-Efficient Cache Design. *Proc. Int'l Conf. on Parallel Arch. and Compilation Techniques*, pp. 61-70, September 2001.