Reducing Static Power Dissipation in Region-Based Caches

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

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

Abstract: *The power consumption of the memory system is a major concern for computer architects since on-chip caches take up a significant portion of the die area. Separate methods have been proposed to alleviate both static and dynamic power consumption in the caches. Region-based caching is a horizontal partitioning technique that reduces dynamic power by adding small caches next to the L1 data cache to hold references to the stack and global regions. Drowsy caches reduce leakage power, a growing concern in advancing circuit technologies, by keeping inactive lines in a low-power mode until they are accessed. We propose merging the two techniques to reduce the overall power consumption of the caches, and show that the combination of the two can be more beneficial than the sum of their parts. Running programs from the MiBench benchmark suite, we obtain a maximum power reduction of 70% with a slowdown of only 1%.*

1 Introduction

Power has always been a major concern in the design of embedded systems and mobile computing devices. As these systems have become more prevalent, the issue has gained more attention. The memory subsystem is a major source of power consumption in embedded processors, consuming over 40% of the overall power in some cases because the on-chip caches comprise such a large part of the die area. [10] Designers have therefore focused a significant effort on reducing power in the memory system, focusing on both dynamic and static power consumption.

Efforts at reducing dynamic power have focused primarily on partitioning the cache into smaller, lower-power structures. [4][13][8] Region-based caching [9] is one such technique. Region-based caching exploits the locality in the stack and global data regions by directing accesses to those regions into separate structures. Using these smaller caches reduces the average power per data access, as in most partitioning techniques. However, region-based caching distinguishes itself from other such methods because it does not increase execution time. The high temporal and spatial locality in the stack and global regions allows for high hit rates even in the smaller caches.

To address the static power consumption of the memory system, a substantial amount of work has focused on reducing leakage power. [12][6][15] As feature sizes shrink, leakage power consumes a larger fraction of the overall processor power. This trend is particularly troubling in caches, because most of the lines in a cache are inactive at any given time and therefore dissipate only static power. Projections show that leakage power may constitute as much as 70% of the total power consumed in caches in a 0.07 micron process. [7] One promising technique for leakage reduction is drowsy caching [3] using a form of dynamic voltage scaling. In a drowsy cache, the supply voltage to inactive lines is reduced, thus lowering their static power dissipation. On an access, the supply voltage must return to its original value before the access is satisfied. Lines are moved from the active state to the drowsy state after a certain number of cycles; depending on the policy used, lines accessed within that interval may remain active. Drowsy caches do not save as much power as other leakage reduction techniques, but they also do not suffer from the dramatically increased latencies the other methods incur.

The characteristics of region-based caching suggest that each of the three cache structures will have more inactive lines and lines that remain inactive longer than a typical cache might. This statement holds true particularly for the heap cache, which is the largest of the three structures, but only sees 30% of the total memory references. These caches must therefore dissipate a large amount of static power and are prime candidates for leakage reduction techniques. In this paper, we propose using drowsy caching techniques to reduce the static power consumption in a region-based caching system. We also show that the characteristics of region-based caching allow us to use a more aggressive drowsy caching policy, thus reducing the leakage power further with a minimal performance impact.

2 Background studies

2.1 Region-based caching

Cache partitioning schemes generally fit into two

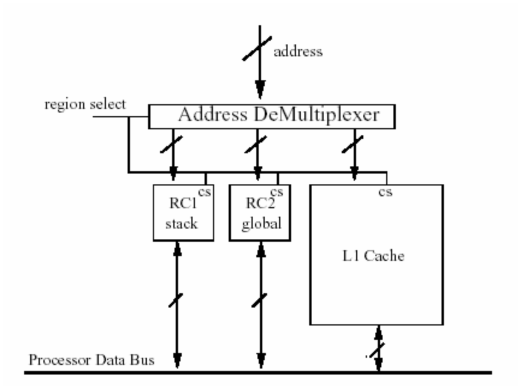

Figure 1: Memory system design for region-based caching

categories: vertical and horizontal partitioning. Since the power dissipated in a cache access is proportional to size of the cache, these techniques reduce power by directing accesses to smaller structures. In vertical partitioning, the cache hierarchy contains an additional level closer to the processor than the L1 cache. Examples include line buffers [4][13] and filter caches [8]. Such structures provide low-power accesses for data with short-term locality, but they typically have low hit rates and require an L1 access after each miss, thus increasing the average L1 latency. In horizontal partitioning, cache lines are split into smaller segments, and the processor accesses only the segment that is referenced. Since only the referenced segment draws power, horizontal partitioning schemes provide another way to reduce power consumption. Cache sub-banking [4][13] is an example of one such technique.

Region-based caching [9] is another form of horizontal partitioning that aims to exploit the high locality of stack and global data. As shown in Figure 1, separate caches are added to the L1 data cache for stack and global data. Accesses to those regions are directed to the appropriate caches; all other data accesses are routed to the L1 data cache. On a memory reference, only the appropriate region cache is activated and draws power.

There are several benefits to region-based caching. Because the working sets of the stack and global regions are relatively small, their respective caches can be small as well and therefore dissipate less power on a hit. Since approximately 70% of the references are in those two regions, the dynamic power consumption drops significantly when comparing region caches to a standard L1 data cache. Also, splitting the references among three separate caches reduces the number of conflicts, since there are no longer any conflicts between regions. Each of the caches can therefore be implemented with lower associativity, making them less complex and faster to access.

For all of our region-based caching simulations, we use a modified version of the SimpleScalar ARM model [1]. Dynamic power is modeled using Wattch [2], while the static power consumption is calculated using HotLeakage [14], a tool we discuss in more detail in the next section. In all cases, the processor configuration is an in-order model similar to the Intel StrongARM SA-110. [10]

Figures 2a and 2b show the results of region-based caching when running a representative subset of benchmarks from the MiBench suite [5] on our simulator. 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 of these caches are direct-mapped. We compare this configuration against three different L1 data cache configurations, as in [9]: a direct-mapped 32 KB cache, a 4-way 32 KB cache, and a 5-way 40 KB cache. Figure 2a shows the power consumption of the caches for each of the benchmarks. On average, region-based caching reduces the dynamic power consumption to 45% of the direct-mapped 32 KB cache's power, 39% of the 4-way 32 KB cache's power, and 33% of the 5-way 40 KB cache's power. It reduces the overall power consumption to 76% of the 32 KB direct-mapped cache's power, 68% of the 4-way 32 KB cache's power, and 57% of the 5-way 40 KB cache's power. Although we do not explicitly show the different components of power consumption, we should note that the leakage power remains roughly constant across the four configurations.

Figure 2b shows the performance impact of region-based caching on these benchmarks. In most cases, we see a slight speedup over the baseline configurations. Since region-based caching reduces the number of conflicts, we see a lower average memory latency. Also, the region caches are faster than the highly-associative caches we use, and they

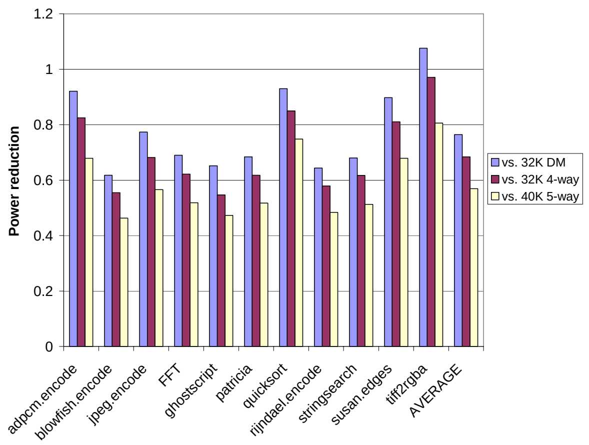

Figure 2a: Effects of region-based caching on power consumption

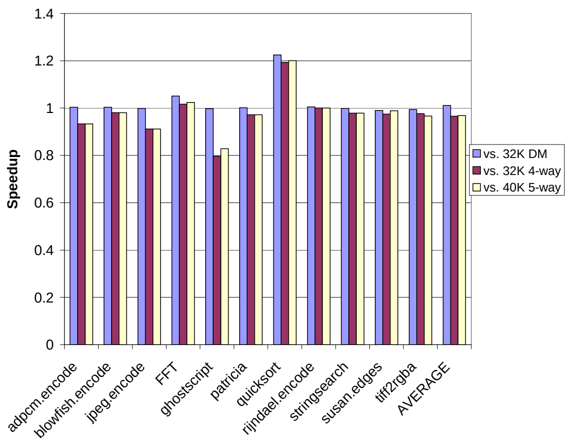

Figure 2b: Effects of region-based caching on performance

provide a greater overall capacity than the 32 KB caches. On average, these benchmarks run about 1% slower with region caches than they do with a 32 KB direct-mapped cache, but they experience a speedup of about 3% over the set-associative caches.

2.2 Drowsy caches

A number of schemes have been proposed to combat the growing cost of leakage power in new circuit technologies. Approaches that use the gated-V_{DD} technique [12][6][15] selectively turn off cache lines containing data that will likely not be reused. Once a line is turned off, its state is lost, and the cost of reloading it from the L2 cache may negate the leakage power savings as well as adversely impact performance. This policy must therefore be implemented conservatively through complex adaptive algorithms that attempt to minimize the costs.

By contrast, drowsy caching is a state-preserving technique; rather than turn off inactive lines entirely, a drowsy cache puts its inactive lines into a low-power mode. When a drowsy line is referenced, that line merely needs to return to full power to satisfy the access. Although this method does not save as much leakage power as a gated-V_{DD} approach, it also does not incur the same performance penalties as gated-V_{DD} does due to its extra L2 accesses. Therefore, drowsy caching policies can be more aggressive at moving lines into a low power state, increasing the potential savings.

Although there are multiple circuit techniques that could be used to implement a drowsy cache, we focus on the dynamic voltage scaling technique proposed in [3]. Figure 3 shows the additional hardware required 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 state of the drowsy bit. The wordline gate insures that a line cannot be accessed while in drowsy mode, thus preventing destruction of the data in that line. When a drowsy line is accessed, the drowsy bit is cleared, returning the supply voltage to its high value. Depending on whether or not the tags are kept active, they may also need to be woken up at this point, potentially increasing the wakeup latency. For a direct-mapped cache, there is no benefit to keeping the tags awake since there is only one line per index; since our models focus on direct-mapped caches in this paper, we assume drowsy tags.

[3] presents two different policies for the setting of the per-line drowsy bits. In the *simple* policy, all lines return to the drowsy state after a certain update interval. By contrast, in the *noaccess* policy, only lines that have not been accessed within the interval return to the drowsy state. The *simple* policy tends to be most beneficial for reducing leakage power, as it more aggressively moves lines from active to drowsy. However, it may increase the execution time in certain cases because it fails to consider locality and may therefore place lines that will be accessed again soon into drowsy mode. The authors show in [3] that this difference in performance is minimal and there is generally little benefit to maintaining information about past accesses. We therefore do not consider the *noaccess* policy in this paper.

To simulate drowsy caches, we integrated HotLeakage [14], a tool for calculating leakage power, into our base simulator. HotLeakage is targeted at architects who wish to consider leakage power in their simulations. Since it contains a detailed model for drowsy caches (used to compare state-preserving and non-state-preserving techniques for leakage control in [11]), it is ideal for our

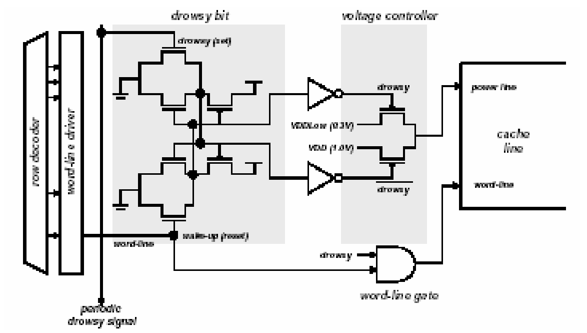

Figure 3: Drowsy cache line

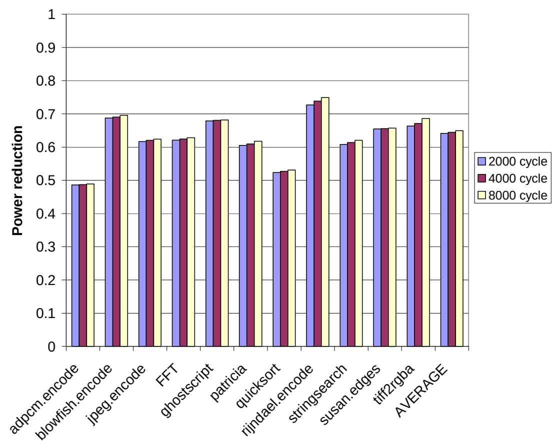

Figure 4: Power reduction of drowsy caching over direct-mapped 32 KB cache for different window sizes

purposes. The drowsy cache model tracks the number of lines in both active and drowsy modes and calculates the leakage power appropriately. It also models the dynamic and leakage power consumption of the additional required hardware: the global counter to track the update interval, the local counters for the *noaccess* policy, and the circuitry added to each line to support a drowsy state.

Figure 4 shows the effects of drowsy caching on power consumption in a direct-mapped 32 KB L1 data cache. As expected, using the smallest update window provides the greatest benefit, reducing the average leakage power to 6.5% and the total power to 64.1% of their baseline values. The larger update windows still provide ample opportunity for reducing power. Using 4000 and 8000 cycle windows, respectively, leakage power is reduced to 7.5% and 8.8% of the base value, and overall power consumption is reduced to 64.5% and 65.0%. Although we do not show the numbers, the performance impact of this technique is very low, measuring less than one-tenth of one percent in all cases.

The dramatic decrease in leakage power seen in these applications is attributable to a couple of factors. First of all, only 26% of all instructions in MiBench reference memory, as compared to 56% of all instructions in the SPEC CPU2000 suite. Secondly, programs running on an in-order core have inherently longer delays between memory accesses than they do when running on an out-of-order machine. The combination of these two factors means that the level of activity in the L1 data cache is extremely low; caches that are frequently inactive benefit significantly from the leakage reduction drowsy caching provides.

3 Current studies

3.1 Combining region and drowsy caches

In Section 2.1, we explored the impact of region-based caching on power consumption. Figure 2a shows that although region-based caching reduces dynamic power consumption, it has little or no effect on static power. The leakage power comprises a significant fraction of the overall power dissipated in the region caches, particularly for the L1 data cache. Because it holds mostly heap data, which exhibits low locality, the L1 data cache must remain relatively large to maintain a high hit rate. Furthermore, less than a third of all data references access this cache. These two factors lead to a large quantity of inactive

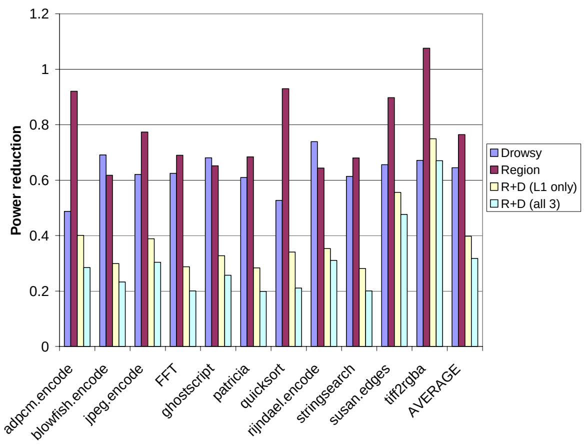

Figure 5a: Power consumption of combined region and drowsy caching

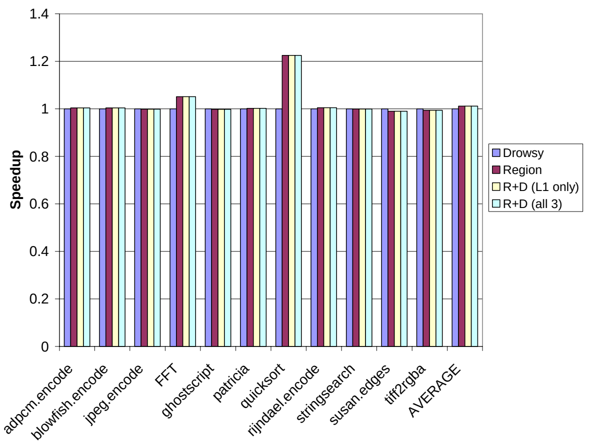

Figure 5b: Performance impact of combined region and drowsy caching

lines in the L1 data cache and therefore high leakage power consumption. The stack and global cache also have a considerable number of inactive lines and dissipate a non-trivial amount of static power as a result.

The above points suggest that region caches are prime candidates for drowsy caching. Since each cache is less active than a standard L1 cache of comparable size would be, it should be able to keep more lines in drowsy mode, thus reducing its static power consumption. We should see the greatest benefit in the L1 data cache, but the stack and global caches should also dissipate less power. In Figures 5a and 5b, we see that this hypothesis is true. The figures show the results of applying the *simple* policy with a 4000 cycle update window to the same region-based caching configuration we used previously. We use two different combinations of region and drowsy caching together: a drowsy L1 data cache with standard stack and global caches, and drowsy caches for all three regions. We normalize all results to a direct-mapped 32 KB L1 data cache. As baselines, we show the numbers for a single L1 data cache with drowsy caching and for our region-based cache configuration without drowsy caching.

Figure 5a displays the power consumption of the combined systems. We see that the combination of region and drowsy caching offers a substantial reduction in the overall power numbers. The total power is reduced to 40% of the baseline value—meaning a 60% reduction in power—if the data cache alone is drowsy, and 32%—a 68% reduction—if all three region caches are drowsy. Drowsy caching alone only reduces the total power to 65%, while region caching alone reduces it to 76%, reductions of 35% and 24%, respectively. We actually see an increase in leakage power over drowsy caching alone when combining the techniques. This result makes sense if the stack and global caches are not drowsy; when they are drowsy, the increased cache capacity leads to a small increase of about 0.4% in leakage power.

We see in Figure 5b that the performance impact of this combination is very small. Since drowsy caching alone has a negligible impact on performance, the runtime penalty is effectively equal to the cost of region caching—about 1%.

The figures show that region and drowsy caching function well together, but do not highlight their ability to perform better as a combination than apart. A quick look at the numbers shows that the reduction in power from the combined techniques is greater than the sum of the reductions for each technique alone—68% versus 59% (35%+24%). The combination has little effect on the dynamic power consumption, which is virtually identical to that of region-based caching alone. We noted above that the combined caches consume slightly more static power because the region caches are larger than the baseline. However, the increase in power is not proportional to the increase in capacity afforded by our region-based caching configuration, so we can surmise that the combination of region and drowsy caching has a positive effect on static power consumption. In fact, merging region and drowsy caching actually allows us to keep lines drowsy longer, thus decreasing the leakage power.

To demonstrate this property, we run each of the benchmarks with a 1 cycle update interval, meaning that each cache line is moved into drowsy mode immediately after it is accessed. We then look at how long each line remains drowsy before it is accessed again. Figures 6a and 6b show this data for the FFT benchmark using a single 32 KB L1 data

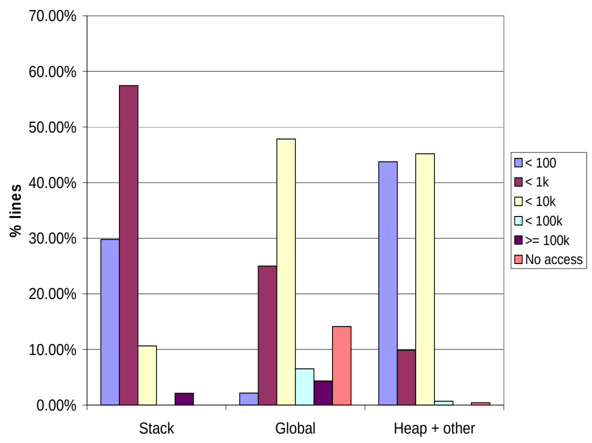

Figure 6a: Drowsy time by region for FFT benchmark using 32 KB direct-mapped L1 data cache

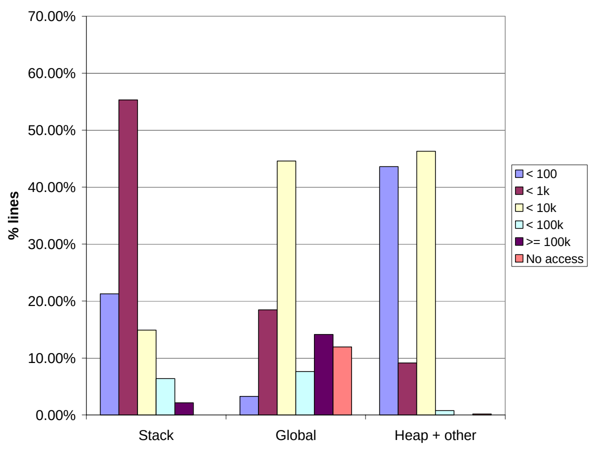

Figure 6b: Drowsy time by region for FFT benchmark using region-based L1 caches

cache and region-based caches, respectively. The data is plotted by region, with each bar showing the percentage of lines in a region that, on average, stay drowsy for the given number of cycles. We split the data into bins, looking at drowsy times ranging from less than 100 cycles up to more than 100,000 cycles. The "No access" group contains lines that are not accessed again once they become drowsy; this set includes those lines that are evicted from the cache before their next access.

The figures validate the point that the region-based caches are able to keep lines drowsy longer than the single cache. In the stack region, we see that the percentages of the smaller intervals, less than 100 cycles and 100 to 1,000 cycles, drop from 30% and 57% to 21% and 55%, respectively. We see a corresponding increase in the larger intervals—lines that are drowsy for 1,000 to 10,000 cycles increase by 4%, and lines that are drowsy for 10,000 to 100,000 cycles increase by 6%. The pattern repeats itself in the other two regions as well. Global data sees a 10% increase in lines remaining drowsy for more than 100,000 cycles. Heap data sees the most modest rise in drowsy time—a 1% increase in lines remaining drowsy for 1,000 to 10,000 cycles and a very small increase in the 10,000 to 100,000 cycle group—because most of the heap data already remains drowsy as long as possible.

3.2 Aggressive drowsy caches

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

Region-based caching already allows us to be more aggressive with our drowsy caching policies because it splits the reference stream and ensures that each cache only sees a small 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 L1 data cache, which tend to rarely be referenced, and still relatively aggressive with the lines in the stack cache once we know they have become inactive.

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 it is accessed—the structure of the drowsy cache line also changes. The drowsy bit is no longer necessary, since each line is either on 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 one might think, as repeated accesses to a line will keep that line active and reduce the number of mode switches.

Figure 7 compares the effects of very small window sizes on the power consumption. We use update

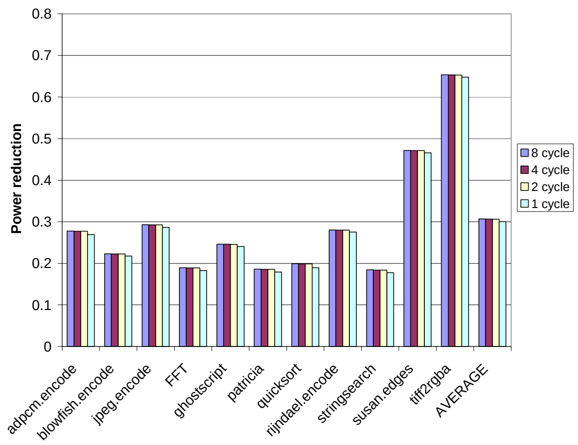

Figure 7: Power reduction for small window sizes

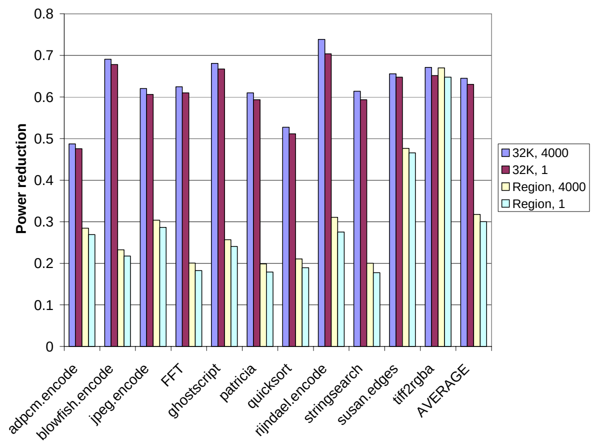

Figure 8: Comparison of large and small windows

windows of 1, 2, 4, and 8 cycles for all three region caches, and normalize our results to a direct-mapped 32 KB L1 data cache. We see that, as expected, reducing the window size almost always reduces the power. The total power consumption is not very different between these different window sizes; the average power reduction of a 1 cycle window is only 0.7% less than that of an 8 cycle window. However, this result is still significant when we consider that shrinking the window size primarily affects the leakage power, which drowsy caching greatly reduces in the first place. On average, the leakage power is 16% less with a 1 cycle window than it is with an 8 cycle window. The overall power consumption for a 1 cycle window is 30% of the baseline value, a reduction of 70%.

Figure 8 highlights the difference in power between typical drowsy caches with a large update window and aggressive drowsy caches in which lines are put into drowsy mode right after they are accessed. We compare a 4000 cycle update window to a 1 cycle update window for both a single L1 data cache and for our typical region-based caching configuration. This graph primarily shows the added benefit of combining drowsy and region caches. When the cache configurations are identical, the total power consumption is similar across window sizes. As noted above, however, the reduced window size has a significant impact on the leakage power. For a 1 cycle window, the leakage power is 31% less than it is with a 4000 cycle window. This decrease in power does not greatly affect the performance of the benchmarks, as both configurations suffer a slowdown of roughly 1%.

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 a while. 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 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 power consumption—at most, 70%—with a minimal performance penalty.

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. We also believe that we can improve the benefits of the combination by maintaining separate update intervals for each of the region caches, allowing us to tailor the aggressiveness of the drowsy caching policy to the needs of each region. We plan to explore these possibilities further in our future work.

References

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

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

[3] K. Flautner, et al. 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*, 1999.

[5] M. R. Guthaus, et al. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *IEEE 4th Workshop on Workload Characterization*, 2001.

[6] S. Kaxiras, Z. Hu, M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. *Proc. 28th Int'l Symp on Comp. Arch.*, 2001, pp. 240-251.

[7] N. S. Kim, et al. Drowsy Instruction Caches: Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction. *35th Annual IEEE/ACM Symposium on Microarchitecure (MICRO-35)*, November 2002, pp. 219-230.

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

[9] 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. (CASES '00)*, pp. 120-127, November 2000.

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

[11] D. Parikh, et al. Comparison of State-Preserving vs. Non-State-Preserving Leakage Control in Caches. *WDDD '03*, June 2003.

[12] M. Powell, et al. 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.

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

[14] Y. Zhang, et al. 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.

[15] H. Zhou, et al. Adaptive mode-control: A static-power-efficient cache design. *Proc. Int'l Conf. on Parallel Arch. and Compilation Techniques*, 2001, pp. 61-70.