Exploiting Temporal Locality in Drowsy Cache Policies

Salvador Petit, Julio Sahuquillo, Jose M. Such
Computer Engineering Departament
Polytechnic University of Valencia
Valencia, Spain
+34 96 387 7570
{spetit, jsahuqui}@disca.upv.es
josucap@fiv.upv.es

David Kaeli
Electrical and Computer Engineering Departament
Northeastern University
Boston, Massachusetts
+1 617 373 5413
kaeli@ece.neu.edu

ABSTRACT

Technology projections indicate that static power will become a major concern in future generations of high-performance microprocessors. Caches represent a significant percentage of the overall microprocessor die area. Therefore, recent research has concentrated on the reduction of leakage current dissipated by caches. The variety of techniques to control current leakage can be classified as non-state preserving or state preserving. Non-state preserving techniques power off selected cache lines while state preserving place selected lines into a low-power state. Drowsy caches are a recently proposed state-preserving technique. In order to introduce low performance overhead, drowsy caches must be very selective on which cache lines are moved to a drowsy state.

Past research on cache organization has focused on how best to exploit the temporal locality present in the data stream. In this paper we propose a novel drowsy cache policy called Reuse Most Recently used On (RMRO), which makes use of reuse information to trade off performance versus energy consumption. Our proposal improves the hit ratio for drowsy lines by about 67%, while reducing the power consumption by about 11.7% (assuming 70nm technology) with respect to previously proposed drowsy cache policies.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles – *cache memories*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids.

General Terms

Performance, Design, Experimentation.

Keywords

Set-Associative Caches, Low-Power, Drowsy Cache Policies, Temporal Locality, Reuse Information.

1. INTRODUCTION

Power has become a major design concern in current microprocessors. The power dissipated has two main components: 1) dynamic power, and 2) static or leakage power. Dynamic power is dissipated due to transistor switching activity, while leakage power continuously dissipates, even when transistors are idle. In current CMOS technologies, dynamic power is dominant and leakage power is less of a concern. However, technology projections for the foreseable future (e.g., 70nm technologies) suggest that static power will dominate overall chip power [13].

Leakage power is dependent on the number of transistors and transistor features implemented. The majority of leakage will come from the largest processor components. Recent research work has focused on reducing power dissipation in different elements of the microprocessor (e.g., in caches [6][5][3][12][1], and arithmetic units [4]).

Cache memories occupy a large percentage of the overall microprocessor die area (e.g., 30% in the Alpha 21264 [7]). This percentage of die area has increased in more recent high performance microprocessors that include large L2 caches. Furthermore, some microprocessors include up to three cache levels on chip (e.g., the Itanium2 [11], which includes a 3 MB L3 cache).

Since caches consume so much chip real estate, we need to find techniques where we can *disable* selected lines. Proposed techniques can be broken down into state preserving [5][9], and state non-preserving [12], depending on if the cache line loses or maintains the information stored. To avoid losing the information, a state preserving solution can place a line in a *low-power state preserving mode*, commonly referred to as *drowsy mode*. Lines in this mode maintain their information, but incur additional cycles to access the information.

Schemes preserving the line state do not lose data; accesses can *hit* either in an enabled or low-powered line. Since a hit in a low-powered line can increase the average data access time, the goal of many schemes tries to insure that a significant fraction of the hits are in enabled lines. The goal is to reduce power while maintaining performance.

Set-associative caches reduce the number of misses, though they have a longer access time and increase power dissipation compared to direct-mapped caches. This increase in power is mainly due to the fact that current microprocessors include as many tag comparators as the number of ways in the cache. There is also extra logic introduced to select the data on a hit. Even given these issues, current microprocessors still implement multi-way set associative caches (e.g., the Itanium2 [11], which provides a 4-way set-associative L1 data cache). Furthermore, the L1 data cache of the Itanium2 has a one cycle hit time.

To arrive at the strategy presented in this paper, we first studied the temporal locality present in a number of applications. To evaluate our strategy, we measured the leakage savings and performance impact of two different versions of our scheme. In our approach, we choose to keep a fixed number of ways of each set in a low-power state-preserving state. Experimental results show that when the L1 data cache maintains one enabled line per set (e.g., the LRU entry in each set), power is reduced dramatically, while only impacting performance slightly. The underlying principle of these simple policies is that they exploit temporal locality to reduce leakage power.

Cache accesses can be bursty during program phases, such that only a few sets are accessed for a long period of time [6]. To benefit from this situation, we propose a new state-preserving cache policy (RMRO) to reduce the leakage current in the data cache. The key idea behind this new policy is to exploit *reuse* information [15], which is used to adaptively disable ways and whole sets, while keeping in mind the intrinsic temporal locality of the workloads.

Using this policy, we will enable zero, one, or two lines per set, while the rest of the lines are disabled. Using this approach, the fraction of low-powered lines exponentially increases with the cache associativity, as do the potential benefits of the scheme. Although our proposal can be applied to any level of the cache hierarchy, in this paper we focus on the performance of a 4-way set-associative first-level data cache. We compare our RMRO policy to a recently proposed policy (SOW) [5], comparing both performance and energy. On average, RMRO saves on leakage energy by 11.7% compared to the SOW policy, while does not adversely impact performance.

The remainder of this paper is organized as follows. Section 2 describes recent research on power reduction. Section 3 presents the range of temporal localities present in the SPEC workloads studied. Section 4 evaluates two simple drowsy policies, as well as our new adaptive drowsy policy. Finally, section 5 presents some concluding remarks.

2. RELATED WORK

We begin by studying the research associated with power reduction on conventional caches. We will also discuss a number of approaches for reducing leakage current.

2.1 Resizing the Cache Geometry

A key issue related to managing power dissipation in caches is that cache utilization varies widely across the different applications and even within a single application. When a given application is running, there can exist long periods during which some parts of the data cache are under-utilized or not touched at all. A number of studies have concentrated on how best to organize a cache for power, focusing on: i) the number of sets, ii) the set associativity, and iii) the line size.

To reduce usage on a set basis, Powell *et al.* [12] proposed a mechanism to identify an application's I-cache requirements dynamically in order to reduce current leakage. The mechanism resizes (increases or decreases) the number of cache sets in order to save energy. Basically the mechanism increases/decreases the number of sets in a given period of time, when the number of misses is lower/higher than a given miss-threshold value. The proposed mechanism uses a variable set *mask* to properly access the corresponding set.

Another approach to reducing power is to change the associativity dynamically. Notice one can disable an entire set by disabling all of its lines. Therefore, reducing the number of sets can be seen as selectively reducing the number of ways in a given set. Albonesi [1] proposed a mechanism to decrease or increase the number of ways depending on both the dynamic power dissipated and the performance degradation suffered during a given period. Flautner *et al.* [5] proposed two policies to reduce leakage power. The first or *simple* policy places all line into a lower-power drowsy mode at fixed time intervals (referred to as the update window). The second or *noaccess* policy differs in that only the non-accessed lines present in the previous window are placed into drowsy mode during the next window. Kaxiras *et al.* [6] proposed to power off the under-utilized lines by using counters. Each line has an associated counter that is incremented at fixed time intervals. Each time a given line is accessed, the counter is reset. If the counter saturates, the line is powered off.

Finally, to reduce cache line size, Chen *et al.* [3] proposed using sublines to better match the low spatial locality present in the data stream. In these caches, the minimum fetch unit is a sub-line (versus a full line). A predictor is used to estimate the number of sub-lines that must be fetched on a miss.

Our work falls into this second group of optimizations since we disable/enable the number of ways in a set to reduce current leakage, and can even disable whole sets in this process.

2.2 Leakage Control for Caches: Vdd and Drowsy Caches

A common technique applied to reduce leakage in caches is to reduce the power supplied. Two main methods have been proposed in recent literature are gated-V_{DD} and drowsy caches.

The gated-V_{DD} [12] technique proposed by Powell et al. uses a transistor to gate the supply of the cache SRAM cells. This technique dramatically reduces the current leakage since selected lines are powered off. Of course, when lines are powered off, the stored information is lost. The main drawback of this technique is that L1 cache misses involve additional accesses to the L2 cache, which incur additional energy dissipation.

Drowsy caches [5] are an alternative technique proposed by Flautner *et al*. This technique saves energy by reducing the power supply for the selected cache lines, though the supply is not gated. Proceeding in this way, the technique avoids losing information. These lines are placed in drowsy mode, and need to be awakened prior to accessing the data. The advantage of this technique is that it achieves the same L1 hit ratio as conventional caches; therefore, no additional accesses to the L2 cache are needed. Nevertheless, as the supply voltage is reduced, current leakage is still greater than the leakage introduced using the gated-V_{DD} technique.

The cache designs proposed in this paper deal with drowsy caches, though they can be used for any state-preserving leakage-saving technology. Moreover, they also can be applied with technologies that do not preserve the state, although the modifications to do so are out of the scope of this work.

3. TEMPORAL LOCALITY VARIATIONS

In this section we investigate the temporal locality of lines on a per set basis. The main objective here is to identify those lines possessing low temporal locality. To this end, we measure the percentage of cache line accesses that hit in the most recently used (MRU) line, in the previous most recently used line, and so on.

Experiments were run considering multi-way sets. In particular, we selected a L1 data cache similar to the design used for the Itanium2 (i.e., 16KB, 64B line, 4 way set-associative (64 sets)). Table 1 shows the results. As observed, only one line per set (the most recently used) captures almost 92.15% of all cache hits. If two lines per set are taken into account (half of the cache lines), this percentage increases to 98.45%. Therefore, the remaining half of the cache lines is responsible for 1.55% of all hits. Based on these results, we explored two straightforward, simple and cost-effective policies for drowsy caches: called the MRO and the TMRO policies.

Table 1 – Percentage of hits along the lines of each set.

Benchmark	MRU line	2nd MRU line	Rest
168.wupwise	96.52	3.05	0.43
171.swim	95.80	4.10	0.10
172.mgrid	88.99	8.85	2.16
173.applu	94.05	5.34	0.61
177.mesa	95.25	4.63	0.12
178.galgel	84.10	12.75	3.15
179.art	95.96	3.93	0.11
183.equake	91.86	6.69	1.45
187.facerec	92.93	6.47	0.60
189.lucas	88.53	8.08	3.39
200.sixtrack	86.46	8.87	4.67
301.apsi	95.39	3.34	1.27
Average	92.15	6.34	1.51

The MRO (Most Recently used On) policy, maintains one line as active in each cache set. The obvious choice would be to maintain the most recently used entry, while the remaining lines are placed into drowsy mode. On a hit into a drowsy line, the line becomes the MRU and is woken up to access the data. The previous MRU line is put in drowsy mode, guaranteeing that only one line is awake. Note that switching the MRU line involves dynamic power dissipation; nevertheless, 92.15% of the cache hits do not involve dynamic switching since they hit in the MRU line.

The TMRO (Two Most Recently used On) policy, keeps two lines awake per set (the most recently and the previous most recently used lines). This policy is more conservative than the previous technique, since it doubles the number of lines that are kept awake. Nevertheless, we study this policy because a significant percentage of the cache hits are due to accesses to the second most recently used line. As a consequence, results obtained from TMRO will show less energy savings than MRO, but will incur fewer accesses to drowsy lines.

4. DROWSY POLICIES EVALUATION

4.1 Methodology

Since a drowsy cache must offer a good tradeoff between performance and power dissipation, we evaluate drowsy cache policies and obtain both execution time (IPC) and the power savings. Experiments were run with the HotLeakage simulator framework [16], which models the energy consumption (dynamic and static) on top of the SimpleScalar toolset [2]. On this platform, we run the SPEC 2000 benchmark suite [14] using the MinneSPEC input set [8].

For performance comparison purposes, we model an aggressive out-of-order speculative processor with a two-level cache hierarchy. Table 3 summarizes the baseline system configuration.

For energy comparison purposes, technology parameter values are chosen assuming recent research papers [5][10] and for the valuable help provided by the Hotleakage group at University of Virginia. Table 2 summarizes these values for a 70nm technology.

Table 2 - Energy parameter values.

Energy Configuration	
Technology	TECH_070
Leakage control technique	Drowsy
Time for low to high switch	3 cycles
Time for high to low switch	3 cycles
Low to high switch cost	0.0003
High to low switch cost	0.0001
Extra latency in low leak mode	1 cycle

4.2 Comparing MRO and TMRO with SOW

We compare both MRO and TMRO to the *simple* policy proposed in [5], which we refer to as the Switch Off Window (SOW) policy. Although this work proposes two policies, we only implement one of them because the authors in [3] conclude that the policies achieve similar performance by choosing the appropriate window length. They also conclude that a 4096-entry window length reaches a reasonable compromise between performance and power savings; therefore, we take this value for our experiments.

The tradeoff of saving energy by putting cache lines into drowsy mode is that we will increase the number of cache accesses that hit in drowsy lines. In this section we quantify the current leakage savings for the MRO, TMRO, and SOW policies over a conventional cache, as well as, the percentage of L1 data cache hits in drowsy lines.

As mentioned above, drowsy cache policies offer the same overall L1 cache hit ratio, since they do not introduce additional misses. Nevertheless, they will differ in the percentage of accesses that hit in a drowsy line. Notice that this percentage is directly related to speedup, since the lower this percentage is, the lower the execution time of an application.

Table 3 - Microprocessor core and memory hierarchy configuration.

Microprocessor Core	Issue policy	Out of order
	Instruction fetch queue size	8 instructions
	Branch predictor type	Combined with a 4K bimodal predictor table size and a 12KB history size
	Branch predictor penalty	2 cycles
	Decode, issue, commit bandwidth	4 instructions/cycle
	Register Update Unit (RUU) size	80
	Load/Store Queue (LSQ) size	40
	# of Integer ALU's, multiplier/dividers	4/1
	# of FP ALU's, FP multiplier/dividers	2/1
Memory Hierarchy	Memory ports available (to CPU)	2
	L1 data cache	16KB, 4 way, 64 byte-line
	L1 data cache hit latency	2 cycles
	L2 data cache	1 MB, 2 way, 64 byte-line
	L2 data cache hit latency	11 cycles
	Memory access latency	97,2,2,2

4.2.1 Current Leakage Savings
Figure 1 plots the leakage current consumption (the opposite of leakage savings). Results show that MRO can outperform the baseline by 72%, while TMRO lags the baseline by about 48%, and the SOW policy savings fall in between. As observed, the SOW leakage savings vary widely across different applications, while MRO and TMRO savings remain nearly constant across the applications due to the number of active lines (one and two, respectively). They seem to also remain consistent in execution time. This fact does not consider that some workloads show an irregular behavior that result in high variability in energy consumption (e.g., 179.art).

4.2.2 Decay hit ratio
The *decay hit ratio* is the percentage of all accesses that hit in drowsy lines. The decay hit ratio directly impacts performance, because each hit in a drowsy line invokes a wakeup cycle before the hit can be serviced. Figure 2 plots the decay hit ratio of the three policies.

Results show that the most conservative policy (TMRO) achieves the lowest decay hit ratio, followed by SOW, while MRO presents the highest decay hit ratio.

Further inspecting the results, we can see that the main advantage of the MRO policy is its ability to lower power consumption. The main advantage of the SOW policy is its low decay hit ratio. Although MRO and TMRO are based on the intrinsic behavior of multiple-way set-associative caches, their main drawback is their lack of flexibility. For instance, to improve the decay hit ratio of the MRO policy we need to wake up more lines per set (e.g., two lines per set, as in the TMRO policy). In this case, the decay hit ratio decreases by an additional 6% (see Table 1), but the leakage consumption is nearly two times as large. On the other hand, if one doubles the number of cache sets in order to improve the decay hit ratio, this also doubles the number of lines that are awakened, increasing the current leakage and undermining the main advantages of the MRO policy.

One can conclude that, although given their lack of flexibility, both policies based on temporal locality are simple and cost-effective policies. The MRO policy incurs less current leakage and the TMRO policy obtains the lowest decay hit ratio. The fact of having a quarter (MRO) or the half (TMRO) of the data entries awake, limits our ability to select a single best energy-performance winner. To deal with this tradeoff, we explore variants of the MRO policies, so that lines can be woken up on demand or placed in drowsy mode when they are under-utilized.

4.3 The Reused MRO (RMRO) Policy
The concept of the *reuse* information has been already discussed in the literature [15] to improve cache hit performance. In those applications, reuse history was used to characterize reuse behavior associated with a cache line. The main idea behind this concept is that different L1 *tours* (i.e., the time from when a block is placed into the L1 data cache until it leaves the cache) of the same block will behave homogeneous (i.e., a block's behavior will repeat in time). In this work, we make use of cache line reuse information to improve both performance and energy, across different execution windows.

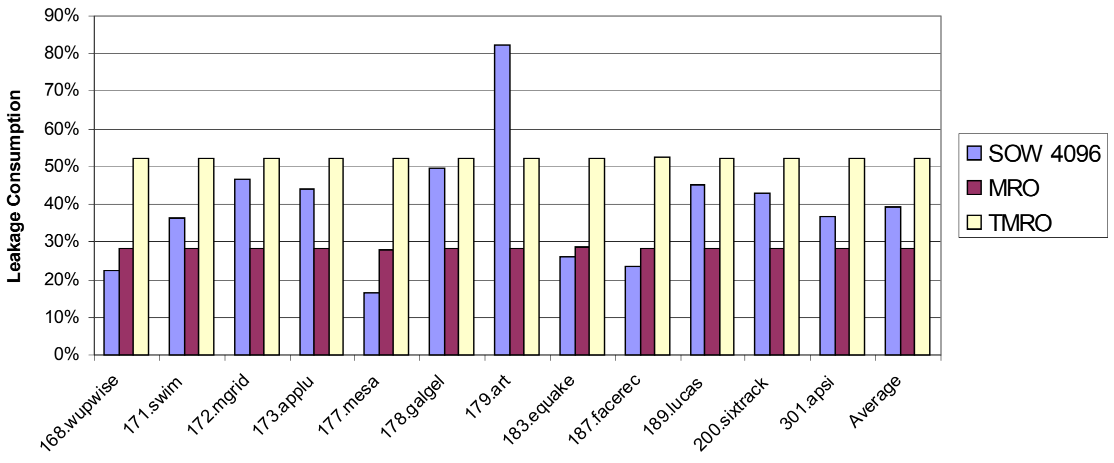

Figure 1 - Leakage energy consumption relative to a conventional cache.

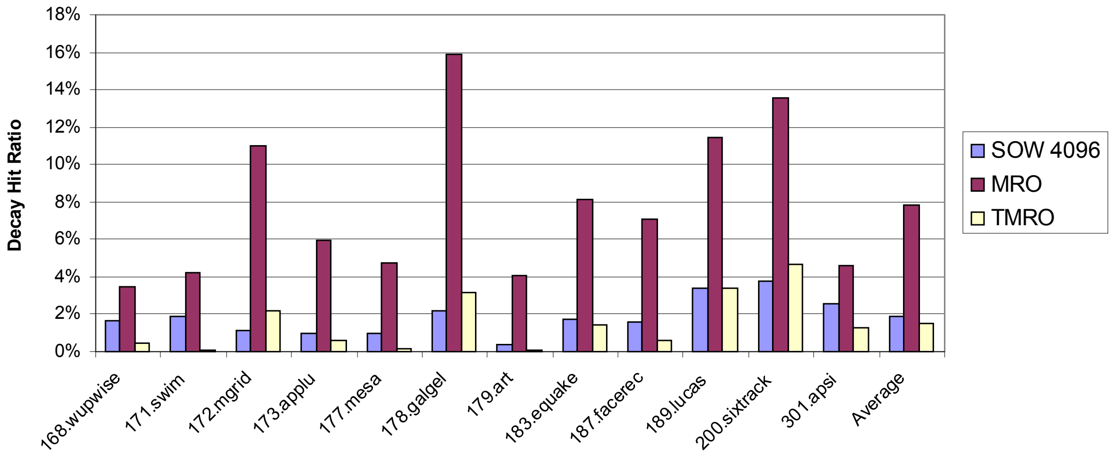

Figure 2 – Decay hit ratio.

The policy we propose, the RMRO (Reused Most Recently used On) policy, saves information about cache line behavior while the line is in a given execution window. This policy wakes up lines on demand. When the execution window expires, the line behavior information is used to choose the line state (i.e., drowsy or awake) for the next execution window. Taking this into account, when the execution windows expires, the policy selects one of the following actions for the next window:

- if no line was accessed during the window, maintain the whole line in drowsy mode.
- if only one line was accessed during the previous window, keep awake only the most recently used line.
- if more than one line was accessed during the previous window, keep awake the two most recently used lines in the set. We bound this number to two based on the results already discussed for the TMRO policy.

4.3.1 Reuse Information

To check how reuse information behaves across windows, we quantify the probability of accessing *n* lines in the next window$_{i+1}$, $n \in [1,4]$ after accessing *m* lines in the current window$_i$. Table 4 shows the results for a 1024 cycle window length. As observed, the main diagonal shows the largest percentage for each row, with the only exception of *n=4*. For instance, the probability of accessing no lines during the next window after accessing no lines during the current window, is about 67%. Therefore, this shows that a line's behavior can remain homogeneous across different time windows.

Table 5 shows the weight of each given probability over the total amount, for instance, the probability of referencing no lines in a given window is 24.6%. Notice that the probability of accessing the full set (4 ways) in two consecutive windows, which was the exception mentioned in the previous table, represents a negligible 1.2% increase over the total amount.

Table 4 - Probability to access *m* lines along the next window after accessing *n* lines in the current window$_i$.

Accessed lines in window i	Accessed lines in window i+1					
	0	1	2	3	4	%
0	67.2	24.5	6.8	1.3	0.2	100
1	18.2	56.6	19.8	4.7	0.8	100
2	6.2	26.5	48.1	15.3	3.9	100
3	2.4	13.6	32.9	39.0	12.1	100
4	1.1	7.4	25.0	35.5	31.1	100

Table 5 - Overall probability to access *m* lines along the next window after accessing *n* lines in the current window$_i$.

Accessed lines in window i	Accessed lines in window i+1					
	0	1	2	3	4	%
0	16.5	6.0	1.7	0.3	0.1	24.6
1	6.2	19.2	6.7	1.6	0.3	33.9
2	1.6	6.8	12.3	3.9	1.0	25.6
3	0.3	1.6	3.9	4.6	1.4	11.9
4	0.0	0.3	1.0	1.4	1.2	4.0

4.3.2 Hardware Implementation

A simple hardware control mechanism is designed to handle the reuse information. A total of six bits per sets are used: one per line to indicate that a given line has been accessed during the window (i.e., the l_i), another one to indicate that only one line has been accessed (i.e., the ol_i), and the last one to indicate that more than one line has been accessed (i.e., the ml_i).

The mechanism works as follows: when a window begins, all control bits are reset. Then, when the accessed line is awakened, the corresponding l_i bit and ol_i bits are set. For another access, if the l_i bit is already set, the ml_i bit must be also set to one. When the window expires, the ml_i bit is inspected, and if set, then the two most recently used lines must be maintained awake for the next window. Otherwise, the value of the ol_i bit is used to decide if either the recently used line or no lines should be kept awake for the next window.

4.3.3 Current Leakage Saving and Decay Hit Ratio

In this section we compare the proposed policy (RMRO) versus the SOW policy, since it showed the best tradeoff between performance and energy consumption.

Notice that the decay hit ratio offered by the proposed policy will vary with the window length. For comparison purposes we look at the window lengths that offered a decay hit ratio close to the one achieved by SOW. Figure 3 shows the results for window sizes of 512 and 1024. As observed, SOW falls in between 512 and 1024, being closer to 1024; therefore, this size will be the used for measuring the leakage consumption.

Since the RMRO has a better decay hit ratio than SOW, one could expect that the leakage savings of RMRO would be less than the results obtained for SOW. However, as Figure 4 shows, the SOW leakage consumption (39.32%) is, on average, about 11.7% higher than the consumption of the RMRO policy (35.21%).

Finally, execution time of drowsy caches has been also simulated, and our results are consistent with those presented in [5], since the performance impact is never more than 1%.

5. CONCLUSIONS

Drowsy caches place selected lines into low-power drowsy mode according to a given drowsy policy. A successful policy should reduce the leakage current while maintaining cache performance. In order to sustain performance, drowsy caches must infrequently access drowsy lines.

In this paper we investigated the temporal locality of cache lines in a multi-way set-associative cache. We then proposed new drowsy policies and compared them against a recently proposed drowsy cache policy (SOW). Our experiments showed that two simple policies which maintain a single (MRO) or two (TMRO) awake lines per set offer power savings of about 72% and 48%, as compared to a conventional cache. The sustained hit ratio on awake lines was 92.15% and 98.49%, respectively. The main drawback of these policies is their lack of flexibility, since they cannot adapt to cache usage. For instance, the power saving with respect to a conventional cache will remain constant for any cache size. This suggests we should explore a policy that dynamically adapts to the workload behavior.

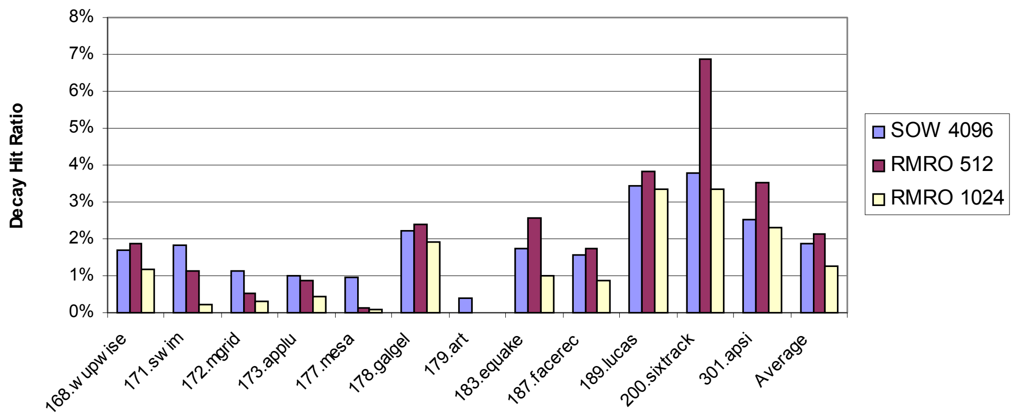

Figure 3 - Decay Hit Ratio.

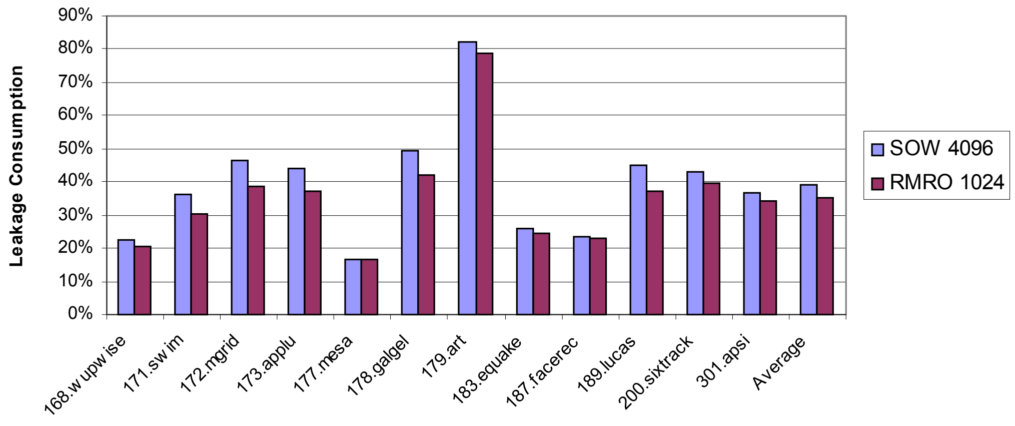

Figure 4 - Leakage Consumption.

In this paper we have proposed an adaptive RMRO policy that makes use of *reuse* information to improve the performance versus energy tradeoff. Reuse information has been used previously improve cache performance, but, to the best of our knowledge, this is the first time that it has been used to improve energy. Our results show that this information can be effectively exploited to improve energy, while maintaining cache performance. Moreover, the proposed policy improves the SOW policy both in performance and in energy savings. For future work, we plan to explore leakage savings in the complete cache hierarchy.

6. ACKNOWLEDGMENTS

This work has been supported by the Generalitat Valenciana under grant GV04B/487. Authors wish to thank the Department of Electrical and Computer Engineering at Northeastern University, Boston, USA, for their support of Salvador Petit's stay at the department.

7. REFERENCES

[1] Albonesi, D. H. Selective Cache Ways: On-Demand Cache Resource Allocation. *Journal of Instruction-Level Parallelism*, Vol. 2, 2000.

[2] Burger, D. C. and Austin, T. M. The SimpleScalar Tool Set, Version 2.0. *Computer Architecture News*, 25 (3), 1997.

[3] Chen, C., Yang, S-H., Falsafi, B., and Moshovos, A. Accurate and Complexity-Effective Spatial Pattern Prediction. *Proc. of the 10th Intl. Symp. on High Performance Computer Architecture,* Feb. 2004.

[4] Dropsho, S., Kursun, V., Albonesi, D. H., Dwarkadas, S., and Friedman, E. G. Managing Static Leakage Energy in Microprocessor Functional Units. *Proc. of the 35th Intl. Symp. on Microarchitecture*, Nov. 2002.

[5] Flautner, K., Kim, N. S., Martin, S., Blaauw, D., and Mudge, T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. *Proc. of the 29th Intl. Symp. on Computer Architecture*, May 2002.

[6] Kaxiras, S., HU, Z., and Martonosi, M. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. *Proc. of the 28th Intl. Symp. on Computer Architecture*, Jun. 2001.

[7] Kessler, R. E. The Alpha 21264 Microprocessor. *IEEE Micro*, Vol. 19, No. 2, Mar. 1999.

[8] KleinOsowski, A. J. and Lilja, D. J. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *Computer Architecture Letters,* Vol. 1, Jun. 2002.

[9] Li, L., Kadayif, I., Tsai, Y-F., Vijaykrishnan, N., Kandemir, M., Irwin, M. J., and Sivasubramania, A. Leakage Energy Management in Cache Hierarchies. *Proc. of the 11th Intl. Conf. on Parallel Architectures and Compilation Techniques,* Sep. 2002.

[10] Li, Y., Parikh, D., Zhang, Y., Sankaranarayanan, K., Stan, M. R., and Skadron, K. State-Preserving vs. Non-State-Preserving Leakage Control in Caches. *Proc. of the 2004 Design, Automation and Test in Europe Conf.*, Feb. 2004.

[11] McNairy, C. and Soltis, D. Itanium 2 Processor Microarchitecture. *IEEE Micro*, Vol. 23, No. 2, Mar.-Apr. 2003.

[12] Powell, M., Yang, S.-H., Falsafi, B., Roy, K., and Vijaykumar, T. N. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. *Proc. of the 2000 Intl. Symp. on Low Power Electronics and Design*, Jul. 2000.

[13] SIA, International Technology Roadmap for Semiconductors, 2001.

[14] Standard Performance Evaluation Corporation. http://www.spec.org/cpu2000/

[15] Tam, E. S., Rivers, J. A., Srinivasan, V., Tyson, G. S., and Davidson, E. S. Active Management of Data Caches by Exploiting Reuse Information. *IEEE Transactions on Computers*, Vol. 48, No. 11, Nov. 1999.

[16] Zhang, Y., Parikh, D., Sankaranarayanan, K., Skadron, K., Stan, M. *HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects*. Tech. Report CS-2003-05, Dept. of Electrical and Computer Engineering, Univ. of Virginia, Mar. 2003.