State-Preserving vs. Non-State-Preserving Leakage Control in Caches

Yingmin Li†, Dharmesh Parikh†, Yan Zhang‡,
Karthik Sankaranarayanan†, Mircea Stan‡, Kevin Skadron†
†Dept. of Computer Science, ‡Dept. of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22904
{*yingmin,dharmesh,karthick,skadron*}*@cs.virginia.edu,* {*yz3w,mircea*}*@virginia.edu*

Abstract

This paper compares the effectiveness of state-preserving and non-state-preserving techniques for leakage control in caches by comparing drowsy cache *and* gated-V_{ss} *for data caches using 70nm technology parameters. To perform the comparison, we introduce "HotLeakage", a new architectural model for subthreshold and gate leakage that explicitly models the effects of temperature, voltage, and parameter variations, and has the ability to recalculate leakage currents dynamically as temperature and voltage change at runtime due to operating conditions, DVS techniques, etc.*

By comparing drowsy-cache and gated-V_{ss} at different L2 latencies and different gate oxide thickness values, we are able to identify a range of operating parameters at which gated-V_{ss} is more energy efficient than drowsy-cache, even though gated-V_{ss} does not preserve data in cache lines that have been deactivated. We are also able to show potential further benefits of gated-V_{ss} if an effective dynamic adaptation technique can be found. These results debunk a fairly widespread belief that state-preserving techniques are inherently superior to non-state-preserving techniques.

1. Introduction

Power is rapidly become a design constraint not only in the domain of mobile devices but also in high performance processors. Although dynamic power —caused by switching activity—is the major source of total power dissipation in today's process generation, static power—caused by leakage current even when circuits are not switching—is gaining in importance for CMOS designs due to technology scaling. The 2001 International Technology Roadmap for Semiconductors (ITRS) [19] predicts that by the 70nm generation, leakage may constitute as much as 50% of total power dissipation. This makes efforts at leakage control essential to maintain control of power dissipation in both high-performance and mobile/embedded processors.

Recently, a great deal of research work in the architecture community has focused on reducing leakage power in the caches, e.g. [7, 10, 11, 13, 18, 20, 22], branch predictor [12], register file [1], issue queues [5, 8], and the ALUs [6]. Leakage control at the architecture level is attractive, because architectural techniques can control large groups of circuits (*e.g.* cache lines, banks, or the entire cache) at once. Leakage control for caches has been an especially active area of study because caches comprise such a large portion of chip area. Recent work [7, 10] has suggested that *state-preserving* techniques are the best choice for leakage control in the first-level (L1) caches, because they do not incur costly accesses to the second-level (L2) cache when reading data that has been placed in low-leakage or "standby" mode.

This paper shows that when the L2 cache offers a sufficiently fast access time (*e.g.*, when the L2 is on chip), *non*-state-preserving techniques can be superior. And even when the L2 is not especially fast, non-state-preserving techniques can still be superior if runtime adaptivity can identify the proper decay interval.

To perform this study, we introduce *HotLeakage*, a new architectural model for subthreshold and gate leakage that has been publicly released on the web. HotLeakage explicitly models the effects of temperature, voltage, and parameter variations, and has the ability to recalculate leakage currents dynamically as temperature and voltage change at runtime due to operating conditions, DVS techniques, etc.

The next section of this paper describes the two leakage-control techniques that we study and the timing and performance assumptions that we make in our simulations, and then Section 3 describes our simulation setup and the benchmarks we use, Section 4 presents the results of our comparison study, and Section 5 concludes the paper.

2. Leakage Control for Caches

The design space for low-leakage caches is daunting, encompassing the choice of size and threshold voltage for each transistor, the row and bitline length, and many more parameters too numerous to mention. Here we focus on just one dimension that can be treated within the scope of a sin-

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)
1530-1591/04 $20.00 © 2004 **IEEE**

gle paper, namely the choice of state-preserving versus non-state-preserving architectural leakage-control techniques in the L1 data cache.

Recent literature has suggested that state-preserving techniques are preferable for leakage control in L1 D-caches, because they do not lose data values and hence do not unnecessarily incur the extra delay and energy associated with reloading that data from the L2 cache. In contrast, our results suggest that this is often not so, that the extra cost of accessing L2 with non-state-preserving techniques is offset by other important factors.

Hanson et al. [10] found that for L1 caches, *reverse body bias* (RBB) or auto-backgate-controlled MTCMOS (ABB-MTCMOS) [16]—a state-preserving technique that manipulates threshold voltages—outperformed gated-V_{ss}, primarily because they used long decay intervals that minimized opportunities for saving energy, and because they did not decay the cache tags [9] (thus avoiding time wasted to waken and read the tags on misses). We have chosen not to study RBB here, both because RBB presents some manufacturing challenges and, more importantly, because recent work by Intel suggests that its effectiveness is limited at future technology nodes by gate-induced drain leakage (GIDL) [14].

Flautner et al. [7] did not directly compare their proposed drowsy-cache scheme against gated-V_{ss}, but suggested that its state-preserving nature is a major advantage.

2.1. Lowering the Quiescent V_{DD} (Gated-V_{ss})

Leakage currents decrease as the supply voltage (V_{dd}) is lowered. The *gated-V_{dd}* structure was introduced as a micro-architecture technique by Powell et al. in [18] as a way to reduce leakage power by using a high threshold "header" transistor to disconnect a cell, row, or way in the cache from V_{dd}. This high-threshold transistor drastically reduces the leakage of the circuit because it breaks the connection to the power supply. While this technique is efficient in saving leakage, there is the disadvantage that the cell loses its state (information). This means that there will be some performance penalty when the data in the cell is accessed and needs to be fetched from a farther level of the cache. This is harmless if the next access to that line would have been an eviction anyway (*true miss*); but if useful data was discarded, the next access will be an *induced miss.* This has important consequences. First and foremost it causes dynamic power dissipation due to an extra L2 access. Second, an induced miss might cause the program to run longer and hence increase total energy consumption. Gated-V_{dd} was proposed in [13] for shutting down individual lines in a cache to save leakage when a line is idle.

There are two possible locations we can place this sleep transistor. The technique with this transistor connected to ground is called gated-V_{ss} while the technique with this transistor connected to V_{dd} is called gated-V_{dd}. Because connecting this transistor to ground will effectively prevent bitline leakage, gated-V_{ss} is more widely used. Connecting this transistor to V_{dd} can save some gate leakage but will induce lots of bitline leakage. According to our experiments, even at small gate oxide thickness values, when the gate leakage is quite large, the savings in gate leakage still can not offset the loss in bitline leakage compared with drowsy cache technique, making gated-V_{dd} less preferable as a non-state-preserving technique . In the following sections, we only investigate gated-V_{ss}.

2.2. Drowsy Caches

An alternative method, proposed by Flautner et al. in [7], achieves significant leakage reduction by putting a cache line into a low-power "drowsy" mode. In drowsy mode, the information in the cache line is preserved by switching its V_{dd} to a separate power supply that is only about 1.5 times the threshold voltage. This reduces leakage current dramatically due to short-channel effects and preserves the value that is stored, making this another *state-preserving* technique. Like ABB-MTCMOS, there is still some overhead because V_{dd} must be returned to the proper level before the value can be safely read. Drowsy caches do not reduce leakage as much as *gated-V_{ss}*, because the cells are not fully disconnected from the power supply. The advantage of drowsy cache is the low penalty of accessing a drowsy line in standby: induced misses do not require an L2 access but only 1-2 cycles to restore the full voltage for that line. Induced misses for drowsy caches might therefore better be called *slow hits*.

2.3. Leakage Energy Modeling

We have developed and released a software model of leakage—based on BSIM3 [2] technology data and the Butts and Sohi abstractions [4]—that is computationally very simple, integrated into popular power-performance simulators like Wattch, extended to accommodate other technology models, and used to model leakage in a variety of structures (not just caches, which are the focus of this paper). We call our model HotLeakage, because it includes the exponential effects of temperature on leakage. Temperature effects are important, because leakage current depends exponentially on temperature, and future operating temperatures may exceed 100° C [19]. In fact, HotLeakage has the ability to recalculate leakage currents dynamically as temperature and voltage change at runtime due to operating conditions, DVS techniques, etc.

HotLeakage has high accuracy because parameters are derived from transistor-level simulation (Cadence tools). Yet like the Butts and Sohi model, simplicity is maintained by deriving the necessary circuit-level model for individual cells, like memory cells or decoder circuits, and then taking advantage of the regularity of major structures to expresse leakage in simple formulas similar to the Butts-Sohi model. All necessary components of this formula are encapsulated in lookup tables.

Based on the BSIM3 v3.2 [2] equation for leakage in a MOSFET transistor, our subthreshold leakage model of a single transistor is given by the following equation:

$$I_{leakage} = \mu_0 \cdot C_{OX} \cdot \frac{W}{L} \cdot e^{b(V_{dd}-V_{dd0})} \cdot v_t^2 \cdot \left(1 - e^{\frac{-V_{dd}}{v_t}}\right) \cdot e^{\frac{-|v_{th}|-V_{off}}{n \cdot v_t}} \quad (1)$$

Butts and Sohi point out that their single k_{design} model is suitable only for cases where the parameters of N and P

transistors are very close, and otherwise two k_{design}'s are needed. We indeed found that the parameters of N and P transistors differ too much, so HotLeakage applies different k_{design} factors to the N and P transistors, k_n and k_p. This means that for a specific cell, the leakage current is given by the following equation:

$$I_{cell\_leakage} = n_n \cdot K_n \cdot I_n + n_p \cdot K_p \cdot I_p \qquad (2)$$

n_N and n_P are the number of NMOS and PMOS transistors in the cell, and I_N and I_P are the calculated unit leakage current of NMOS and PMOS transistors according to Equation 1.

Our model also includes gate leakage for 70nm technology, where gate leakage becomes dominant. We use AIMspice [17] as the circuit simulator, which includes BSIM4 among the supported models for gate leakage. Gate leakage is strongly dependent on the gate oxide thicknessx (Tox) and supply voltage. It is weakly dependent on the temperature. From the transistor-level simulations, we derived these factors with curve-fitting and incorporated it into our models. The final equation for the gate leakage of one transistor is as following:

$$I_{gateleakage} = (I_{unitleakage} + b\_mos\_t \cdot (T - T_{base})) \cdot \frac{W}{L} \cdot e^{(a\_mos\_vdd(V_{dd} - V_{base}))} \cdot e^{c\_mos\_tox(Tox_{base} - Tox)} \qquad (3)$$

Detailed information about HotLeakage and how to use it appear in a technical report [21].

2.4. Modeling of Cache Leakage Control

We have implemented a generic abstraction for modeling architectural leakage control techniques based on putting individual lines into standby mode, allowing us to study techniques like gated-V_{ss} [13], drowsy cache [7], and reverse-body-bias [16].

Most dynamic leakage-control techniques partition a structure into active and passive portions. This can be done at various granularities; most recent work has done this at the granularity of rows in the SRAM array, which correspond to cache lines.

These leakage control techniques also require some extra hardware that adds to the area of the structure. Hence, these methods have the following costs: dynamic power due to the extra hardware, leakage power due to the extra hardware, dynamic power due to mode transitions and dynamic power due to extra execution time.

The energy benefit of the techniques we have described is the leakage power saved in the lines that are in standby mode. This saving is proportional to the average percent area that is kept in standby mode (the *turnoff ratio*). Our experiments compute a *net* energy savings that subtracts from this gross benefit the costs itemized above: Wattch automatically capatures the extra energy due to longer runtime (item #4 above); this is compared to the energy from a baseline simulation with no leakage control, and the resulting cost is added to the other costs itemized above (#1–3). These are then subtracted from the gross leakage savings. For both techniques, we use a global counter that counts from zero up to one-fourth the decay interval (defined as *update window size* in [7]) and then starts over. This global counter is incremented every cycle. Following [13], each line uses a local two-bit counter; when the global counter rolls over, all two bit counters are incremented. When a line is accessed, its local counter is reset. When a two-bit counter reaches its maximum, the line has been idle for the full decay interval, it is assumed that the line's usefulness has decayed, and the line is deactivated. In the original drowsy-cache paper, this corresponds to the *noaccess* policy. To be fair, we used this policy for both gated-V_{ss} and drowsy.

For both techniques, we decay the tags too (defined as *drowsy tags* in [7]). Access to a drowsy line in such a case takes at least three cycles due to the need to wake up tags before they can be checked. For gated-V_{ss}, on the other hand, a line in standby mode has no useful information, and tags need not (cannot) be checked. This means that on a true miss to L2 when tags are in standby, gated-V_{ss} is actually faster. Hanson et al. also kept the tags awake in their study [9, 10].

3. Simulation Set-Up

3.1. Processor Model

All simulations were performed with Wattch [3] augmented by HotLeakage. Unless stated otherwise, this paper uses the baseline configuration resembling as much as possible the configuration of an Alpha 21264 [15].

In the original drowsy paper, the L1 data cache used is 32 KB in size and 4-way set associative and the L1 instruction cache is 32 KB in size and direct mapped. Both caches use line size of 32 bytes and a hit latency is one. In contrast, we use 64 KB, 2-way caches with 64 B lines for both, consistent with a 21264.

For Wattch and HotLeakage technology parameters we use values for a 70 nm process at V_{dd} 0.9V and 5600 MHz. It is important to note that, because our Wattch model does not include state-of-the-art power-management techniques that would be expected in the 70nm generation, our estimates for dynamic energy may be pessimistically high, this will exacerbate the energy cost of any performance loss and will tend to favor the drowsy technique, and we feel this factor, if anything, strengthens our results.

3.2. Benchmarks

In our comparative evaluation of various leakage control techniques, we use 11 integer benchmarks from the SPEcpu2000 suite. The benchmarks were compiled for the Alpha ISA and statically linked using the Compaq Alpha compiler (with *peak* settings) For each program, we skip the first two billion committed instructions to avoid unrepresentative startup behavior at the beginning of the program's execution, and then simulate 500 million committed instructions using the reference input set.

4. Results

4.1. L2 Latency

Our results roughly duplicate those in [7]. They report slightly higher leakage savings and slightly lower perfor-

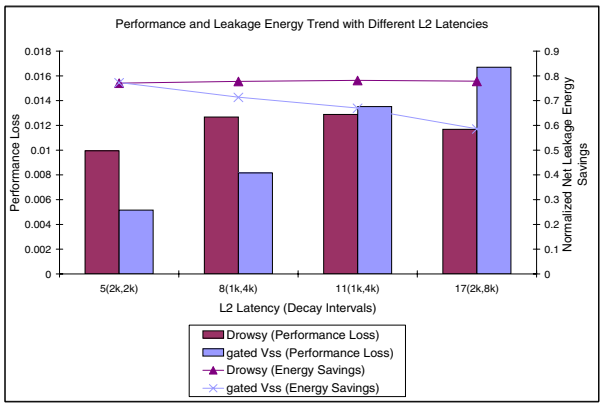

Figure 1. Net leakage savings and performance loss with different L2 latencies

mance loss. The former we attribute to differences in our models, including the different choice of threshold voltage and our use of BSIM3 models. The latter we attribute to our choice of shorter decay intervals that—for our leakage model—we found to give better energy savings.

Figures 1 presents the *net* cache-leakage savings and the performance loss for different L2 cache latencies, from 5 cycles, as might be seen for a fast on-chip L2, to 17 cycles. These values are the average values across the 11 benchmarks we adopted for our experiment. Note that, in order to report a measure that represents the actual "profit" in terms of energy saved, the net savings subtracts the extra dynamic energy expended due to the leakage control scheme from the total reduction in leakage that is realized by deactivating cache lines. The dynamic energy overhead is computed by comparing the total dynamic energy with and without the leakage-control scheme activated. This accounts for the contributions from (and overlap among) (a) activity in the decay counters (gated-V_{ss}), (b) extra L2 accesses (gated-V_{ss}), (c) extra tag accesses (drowsy), and (d) extra runtime.

These results show that for 5 cycle L2 caches, gated-V_{ss} is superior to drowsy cache in terms of both energy savings and performance loss. At 8 cycles, gated-V_{ss} is superior in terms of performance, but drowsy is slightly superior in terms of energy savings. At 11 and 17 cycles, drowsy cache becomes superior for both performance and energy.

Most importantly, these results show that contrary to widespread belief, non-state-preserving techniques are not inherently inferior. There are five reasons for this. First, gated-V_{ss} is able to almost entirely eliminate leakage, whereas state-preserving techniques like drowsy and RBB still exhibit a non-trivial amount of leakage. Second, a well-tuned decay interval will minimize so-called induced misses, misses that result purely from premature deactivation of a line that contains useful data. Third, induced misses are not inherently bad. Even if data remains "live", if its next use is sufficiently far in the future, it may be worthwhile to incur a modest performance loss to save energy that is otherwise expended keeping the data active. Fourth, in an aggressive out-of-order machine, modest L2 access latencies for induced misses can be tolerated. Finally, when tags are decayed, gated-V_{ss} is actually faster on true misses when a line is in standby—which is the more common type of miss. The drowsy technique must first wake up the tags, then check them, only to find that the data is not resident and an L2 access is required. In contrast, gated-V_{ss} can immediately begin checking the tags of active ways, and ways that are in standby are guaranteed to be misses and need not be checked.

For the range of L2 access latencies that are typically observed for on-chip caches, it is therefore false to automatically assume that an L2 access is too costly. Of course, as L2 latency increases, the above factors that mitigate for gated-V_{ss} become less and less helpful. For the longest L2 latency we tested, gated-V_{ss} is no longer able to hide a significant portion of L1 miss times, and the state-preserving nature of drowsy cache becomes a major advantage.

4.2. Temperature

We compare energy savings at 85°C and 110°C with a L2 cache latency of 11 cycles. Because leakage is exponentially dependent on temperature, the energy savings is much higher for both schemes. With drowsy cache technique, the normalized energy saving rises from 72.1% at 85°C to 78.2% at 110°C; with gated-V_{ss} technique, the energy savings changes from 54.9% to 67.0%.

We mentioned previously that gated-V_{ss} is able to almost entirely eliminate leakage, whereas state-preserving techniques like drowsy and RBB still exhibit a non-trivial amount of leakage. As leakage increases with temperature, this advantage for gated-V_{ss} increases too. But this advantage is offset by the fact that the higher leakage at higher temperature makes shorter decay intervals attractive for both gated-V_{ss} and drowsy, and gated-V_{ss} is more sensitive to the smaller decay interval. The former factor benefits gated-V_{ss} for programs like *gcc* and *gzip*, but the latter factor penalizes gated-V_{ss} for *gap* and *twolf*. On average, therefore, temperature has surprisingly little impact on the relative performance of gated-V_{ss} and drowsy.

4.3. Tag Decay

We have only had the opportunity to compare gated-V_{ss} when tags are also placed in standby along with the line of data that is being deactivated. If tags are not placed in standby, drowsy no longer suffers extra penalties for true misses. If one simply uses the same decay intervals but keeps the tags live for the drowsy cache, this will reduce the performance loss exhibited by drowsy but also substantially reduce the energy savings, because tags account for 5–10% of the leakage energy in caches, and this leakage energy can no longer be reclaimed. For gated-V_{ss}, on the other hand, there is no advantage to keeping the tags live unless they are used to facilitate adaptive decay intervals.

4.4. Adaptivity

We show in this subsection how much better both schemes could do if an adaptive scheme were employed to allow the cache-decay mechanism to find the best decay interval for each benchmark. For both drowsy and gated-V_{ss},

	Gcc	Gzip	Parser	Vpr	Mcf	Crafty
Drowsy	1k	2k	4k	2k	1k	4k
Gated-V_{ss}	2k	64k	16k	8k	2k	32k

Table 1. Best decay intervals.

we identify the best decay interval for each benchmark,some of them are listed in Table 1.

Adaptivity primarily benefits gated-V_{ss}, because the best decay intervals vary so widely. This in turn is a function of data-usage patterns and available instruction level parallelism that can be used to hide induced misses. For L2 latency of 11 cycles and temperature 85°C, our experiments show that using the best per-benchmark intervals improves energy savings for gated-V_{ss} by 22% compared with the fixed 2k decay interval, from 55% to 67%, and dramatically reduces performance loss, from about 0.88% to about 0.14%. Energy savings for drowsy cache only improve by about 0.5% and performance loss only improves from 1% to 0.9%.

It is to be expected from the analysis in [7] that adaptivity is not necessary for drowsy cache, because for reasonable intervals, it is fairly insensitive to decay interval. Gated-V_{ss} does not require adaptivity to give attractive benefits for on-chip L2 caches, but performs even better with adaptive decay intervals.

We are aware of three methods so far for providing adaptive decay intervals: using an array of bits to select from multiple possible decay intervals, proposed by Kaxiras et al. [13]; the *adaptive mode control* technique proposed by Zhou et al. [22]; and the formal feedback-control technique proposed in [20].

4.5. Thickness of Gate Oxide

Subthreshold leakage energy and gate leakage energy are both highly dependent on the thickness of gate oxide (Tox). For an L2 latency of 5 cycles and 110°C temperature, Figure 3 shows the normalized leakage energy savings for different leakage saving techniques when Tox varies from 7Å to 13Å (Tox less than 10Å seems unlikely according to ITRS and is only shown to illustrate how gate leakage affects the tradeoff between drowsy cache and gated-V_{ss}). Figure 2 shows three kinds of absolute leakage energy values: total leakage energy, subthreshold leakage energy and gate leakage energy. The values in Figure 2 are for cache data cells and tags, not including control periphery circuits like decoders, comparators and amplifier sensors: cache leakage saving techniques we investigate do not save leakage energy for these circuits. From Figure 2 we can conclude that when Tox increases, subthreshold leakage increases while gate leakage decreases.

As we can see from Figure 3, for small Tox value like 7Å, the drowsy cache technique saves more leakage energy. That is because gated-V_{ss} consumes more gate leakage in the cache cells and when Tox is small, gate leakage is much bigger than subthreshold leakage. However, if we exclude the gate leakage from the total leakage, the gated-V_{ss} technique outperforms drowsy. This indicates that if we can save

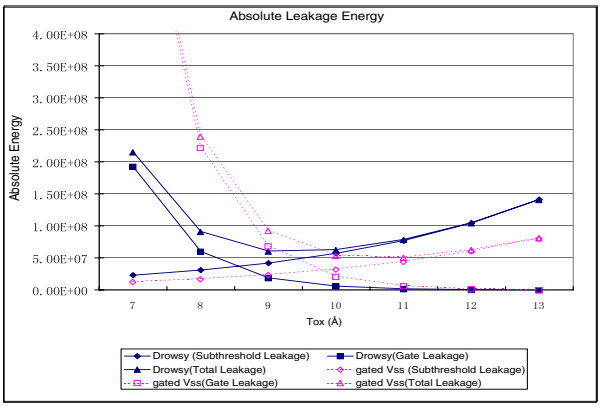

Figure 2. Absolute leakage energy for different Tox

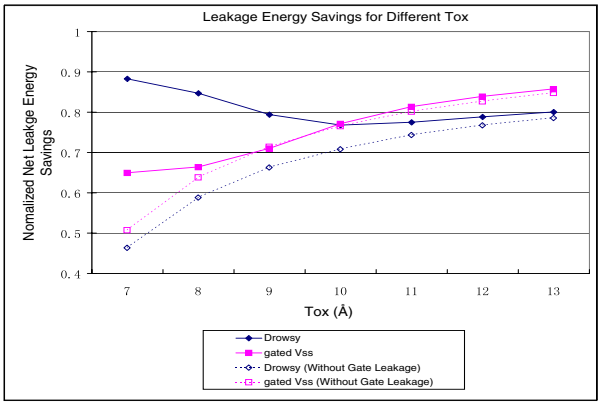

Figure 3. Leakage energy savings for different Tox

gate leakage energy for the gated-V_{ss} technique when the cache line is in low-leakge mode, gated-V_{ss} can still be better than drowsy cache for small Tox values. This might be accomplished by adding a gated-V_{dd} header in combination with the gated-V_{ss} footer or with a new gate-oxide material. For large Tox values, gated-V_{ss} saves more energy. In this case subthreshold leakage dominates the whole leakage energy, and gated-V_{ss} saves much more subthreshold leakage energy than drowsy cache.

5. Conclusions and Future Work

HotLeakage provides the first publicly available microarchitecture level leakage modelling software of which we are aware. Its most important features are the explicit inclusion of temperature, voltage, gate leakage, and parameter variations. HotLeakage provides default settings for 180nm through 70nm technologies (based upon BSIM3 and BSIM4 models) for modelling cache and register files, and provides a simple interface for selecting alternate parameter values and for modelling alternative microarchitecture

structures. HotLeakage also provides models for several extant cache leakage-control techniques, with an interface for adding further techniques. The HotLeakage tool, with all the supporting documents, is available on the web at www.cs.virginia.edu/hotleakage

Using HotLeakage and Wattch, we have compared a state-preserving technique (drowsy cache) against a non-state-preserving technique (gated-V_{ss}). Conventional wisdom holds that the state-preserving technique must be superior, because it incurs less performance loss on access to a line that is in standby mode. In contrast, we have found that at 70nm and for the particular range of parameters we studied, the non-state-preserving technique is actually superior for a set of faster L2 cache latencies that might be seen with on-chip L2s. This remains true when gate leakage is taken into account. The main reasons for this are that gated-V_{ss} reduces leakage by a greater amount than drowsy cache, that the latency to fetch data from L2 when accessing a line in standby mode can be hidden to a significant extent by ILP, and that drowsy cache actually incurs a larger performance penalty than gated-V_{ss} for the more common case of a true (rather than an induced) miss. In addition, the effectiveness of gated-V_{ss} can be expanded by using adaptive decay intervals. As Tox is reduced, gate leakage does reduce the benefit of gated-V_{ss}, and at extremely small Tox values, drowsy cache has an advantage. This scenario is unlikely, however, and can be rectified by using both header and footer on the SRAM cell to gate both V_{dd} and V_{ss} or possibly with a new gate oxide material.

The design space for power-efficient caches is notoriously complex, and even the design space for just these two techniques is too rich to fully explore in this paper. The proper choice of leakage-control technique will depend on a variety of factors, and we hope that the comparison here illustrates some important tradeoffs to consider. The main point that we wish to convey with this work is to debunk the perception that non-state-preserving techniques are inherently inferior. Design of low-leakage caches requires non-state-preserving techniques like gated-V_{ss} to be considered as potentially the most energy-efficient and highest-performance solution.

Acknowledgments

This work was funded in part by the National Science Foundation under grant nos. CCR-0133634, CCR-0105626, and MIP-9703440, a grant from Intel MRL, and an Excellence Award from the University of Virginia Fund for Excellence in Science and Technology.

References

[1] A. Alvandpour, R. Krishnamurthy, K. Soumyanath, and S. Borkar. A low-leakage dynamic multi-ported register file in 0.13um CMOS. In *Proc. ISLPED 2001*, pages 68–71, Aug. 2001.

[2] U. C. Berkeley. BSIM3v3.1 SPICE MOS device models, 1997. `http://www-device.EECS.Berkeley.EDU/bsim3/`.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. ISCA-27*, pages 83–94, June 2000.

[4] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proc. Micro-33*, pages 191–201, Dec. 2000.

[5] A. Buyuktosunoglu, D. H. Albonesi, P. Bose, P. W. Cook, , and S. E. Schuster. Tradeoffs in power-efficient issue queue design. In *Proc. ISLPED 2002*, pages 184–189, Aug. 2002.

[6] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing static leakage energy in microprocessor functional units. In *Proc. Micro-35*, pages 321–32, Nov. 2002.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proc. ISCA-29*, pages 147–57, May 2002.

[8] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proc. ISCA-28*, pages 248–59, June. 2001.

[9] H. Hanson. Personal communication. May. 2003.

[10] H. Hanson et al. Static energy reduction techniques for microprocessor caches. In *Proc. ICCD 2001*, pages 276–83, Sept. 2001.

[11] S. Heo, K. Barr, M. Hampton, and K. Asanović. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proc. ISCA-29*, pages 137–47, May 2002.

[12] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi. Applying decay strategies to branch predictors for leakage energy savings. In *Proc. ICCD 2002*, pages 442–45, Sept. 2002.

[13] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proc. ISCA-28*, pages 240–251, July 2001.

[14] A. Keshavarzi et al. Effectiveness of reverse body bias for leakage control in scaled dual Vt CMOS ICs. In *Proc. ISLPED 2001*, pages 207–12, Aug. 2001.

[15] R. E. Kessler, E. J. McLellan, and D. A. Webb. The Alpha 21264 microprocessor architecture. In *Proc. ICCD 1998*, pages 90–95, Oct. 1998.

[16] K. Nii et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proc. ISLPED 1998*, pages 293–98, Aug. 1998.

[17] Aim-Spice Home Page. http://www.aimspice.com.

[18] 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. In *Proc. ISLPED 2000*, pages 90–95, July 2000.

[19] SIA. *International Technology Roadmap for Semiconductors*, 2001.

[20] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proc. WMPI-2*, May 2002.

[21] 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, U.Va. Dept. of Computer Science, Mar. 2003.

[22] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proc. PACT 2001*, Sept. 2001.