# Comparison of State-Preserving vs. Non-State-Preserving Leakage Control in Caches*

Dharmesh Parikh‡, Yan Zhang†, Karthik Sankaranarayanan‡, Kevin Skadron‡, Mircea Stan†
†Dept. of Electrical and Computer Engineering, ‡Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904
{*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 use "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, 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.*

*This paper duplicates some of the findings of both the drowsy-cache and "cache-decay" papers, but also debunks 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) [27] 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 [11, 14, 15, 19, 25, 31, 33], branch predictor [16, 17], register file [2], issue queues [7, 8, 12, 24], and the ALUs [10]. 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 [11, 14] 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 use *HotLeakage* [32], 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 provides an overview of the HotLeakage model. Section 4 describes the rest of our simulation setup and the benchmarks we use, Section 5 presents the results of our comparison study, and Section 6 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 single 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.

*This paper appeared in the 2003 Workshop on Duplicating, Deconstructing, and Debunking (WDDD), 8 June 2003, in conjunction with ISCA-30.

Hanson et al. [14] found that for L1 caches, *reverse body bias* (RBB) or auto-backgate-controlled MTCMOS (ABB-MTCMOS) [23]—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 [13] (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).

Flautner et al. [11] 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 microarchitecture technique by Powell et al. in [25] 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 [19] for shutting down individual lines in a cache to save leakage when a line is idle. Because the sleep transistor is more effective as a "footer" on the connection to ground–it is easier to prevent bitline leakage this way–this technique is better called *gated-$V_{ss}$*.

## 2.2 Drowsy Caches

An alternative method, proposed by Flautner et al. in [11], 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 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 Modeling of Cache Leakage Control

We have implemented a generic abstraction for modeling leakage control techniques based on putting individual lines into standby mode, allowing us to study techniques like gated-$V_{ss}$ [19], drowsy cache [11], and reverse-body-bias [23].

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:

1. Dynamic power due to the extra hardware
2. Leakage power due to the extra hardware
3. Dynamic power due to mode transitions (active to standby and vice-versa)
4. Dynamic power due to extra execution time, resulting either from extra latency in accessing the structure or extra latency in fetching data from the L2 cache.

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 [11]) and then starts over. Following [19], each line uses a local two-bit counter; when the global counter reaches its maximum value, all two bit counters are incremented. 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. Drowsy cache also proposes the *simple* policy, which uses no per-line access history but rather automatically turns off all lines every $N$ cycles. The simple policy loses out in performance compared to the noaccess policy, but saves more leakage power. The difference seems modest for drowsy due to the fairly low cost of any extra slow hits: there is some increase in performance loss, but also more energy savings. To be fair to both gated-$V_{ss}$ and drowsy, we used the same policy involving counters, namely noaccess.

For both techniques, we decay the tags too (defined as *drowsy tags* in [11]). 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 [13, 14].

A few other simulation details are worth mentioning. The time taken for a line to go to a low-leakage mode from high-leak normal mode (*settling time*) and vice versa was found from circuit simulation and is given in Table 1. Also, for both leakage saving techniques we use the same values of threshold voltage for all

| | Drowsy | Gated-$V_{ss}$ |
|---|---|---|
| Low leak mode to high | 3 | 3 |
| High leak to low | 3 | 30 |

**Table 1. Settling Time.**

the transistors of the same type for a memory cell. In contrast, drowsy uses high-$V_t$ for the access transistors. Modeling this is easy with HotLeakage. But for making fair comparison, we use the same threshold voltages (for 70nm we use 0.190 V for N-type and 0.213 V for P-type transistor). It is true that high-$V_t$ access transistors help drowsy more than gated-$V_{ss}$. High-$V_t$ access transistors only help gated-$V_{ss}$ when lines are awake, while they help drowsy in both situations. But since the bulk of the leakage is when awake, we felt that using the same $V_t$ was the best solution. Finally, HotLeakage models inter-die variation. We use the following three-sigma values for 70nm technology. The values were obtained from [22].

- Length of the transistor: 47%
- Thickness of the gate oxide: 16%
- Supply voltage: 10%
- Threshold Voltage: 13%

The simulator currently models leakage control in caches using the above costs and benefits. The dynamic power calculations are performed using Wattch routines—see Section 4 for details. The leakage power is calculated using our model as configured by the command line options—see Section 3 for details.

# 3 An Accurate Leakage Model for Architects

Although architectural control of leakage energy has been an active area of research in recent years, many of these studies use only abstract models of leakage that do not fully account for all effects that may impact leakage, like supply voltage and temperature; and other studies use circuit-extracted parameters that are not easily incorporated into other researchers' models. Unlike for dynamic power, where widely-available simulators like Wattch [5] have enabled a widespread body of research, there is no widely available model for leakage power. This inhibits leakage research and leads to approximate experiments. Although Butts and Sohi [6] propose a simple model for use at the architecture-simulation level of abstraction, no corresponding software is available. Most importantly, their model cannot easily model leakage when temperature, supply voltage, or threshold voltage vary dynamically: a new "normalized leakage" and $k_{design}$ must be calculated for every possible value. This is inconvenient although feasible for leakage-control schemes like drowsy cache that uses two supply voltages, but intractable for any leakage studies that account for dynamically varying temperature or involve dynamic voltage scaling. Unlike the Butts and Sohi model, we find that $k_{design}$ does in fact vary with temperature, supply voltage, threshold voltage, and channel length. Detailed plots can be found in [32].

We have developed and released a software model of leakage—based on BSIM3 [3] technology data and the Butts and Sohi abstractions—that is computationally very simple, can easily be integrated into popular power-performance simulators like Wattch, can easily be extended to accommodate other technology models, and can easily be 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 [27]. In fact, HotLeakage also includes the heretofore unmodeled effects of supply voltage, gate leakage, and parameter variations.

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.

We hope that this new leakage model and its public availability will facilitate greater research on techniques for controlling leakage power at the architecture level. HotLeakage is publicly available for download at `http://lava.cs.virginia.edu/HotLeakage`. It is a separate library with minimal dependence on the details of SimpleScalar and Wattch, so porting HotLeakage for use with other simulators should be straightforward. We encourage not only such ports, but also any other modifications or extensions users might wish to add.

## 3.1 Subthreshold Leakage

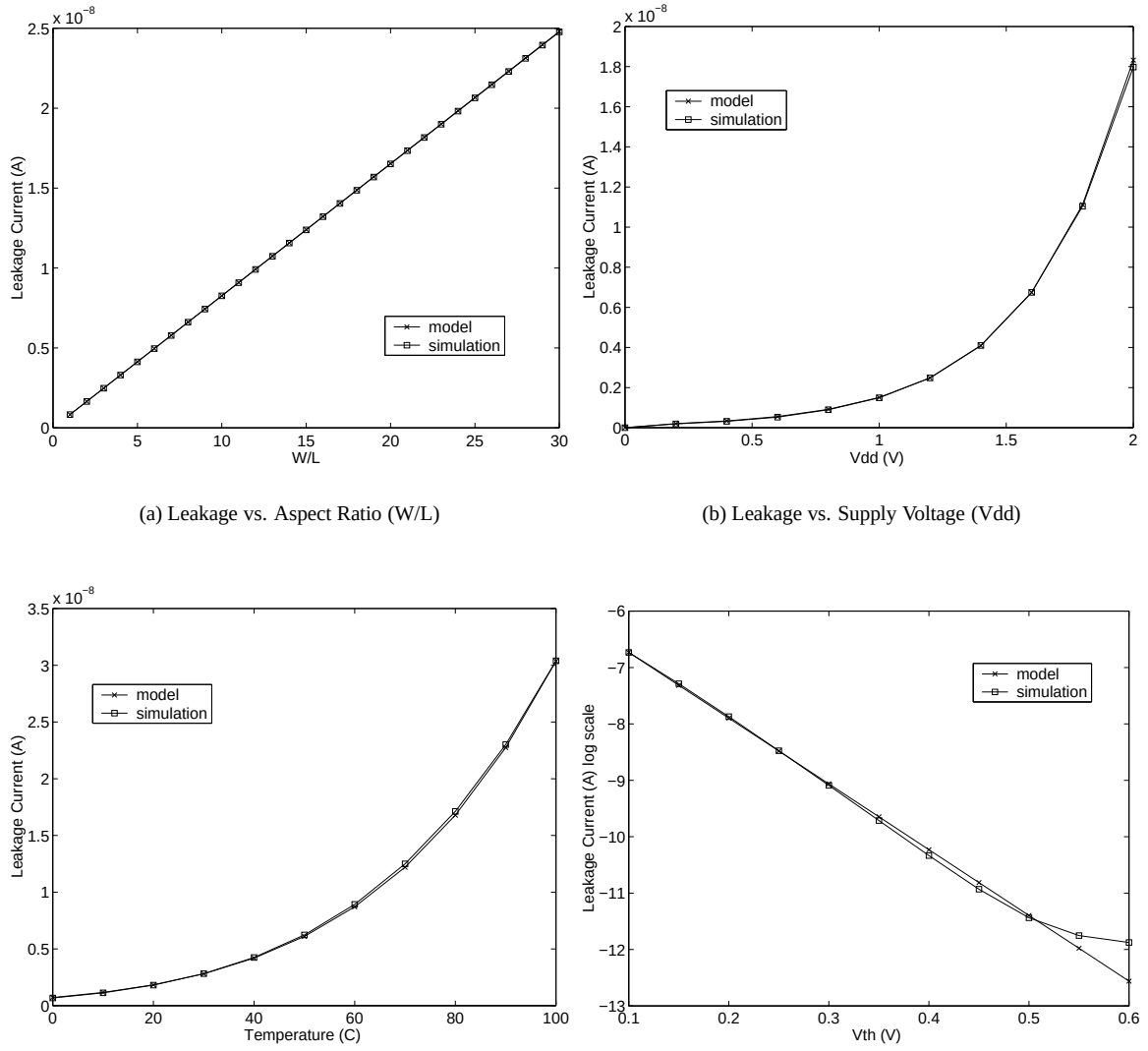
Leakage current is influenced by the threshold voltage, channel physical dimensions, channel/surface doping profile, drain/source junction depth, gate-oxide thickness, $V_{dd}$, temperature, and variations in all these parameters. For long-channel devices, the leakage current is dominated by leakage from the drain-well and well-substrate reverse-bias pn junctions. For short-channel transistors, because of the low threshold voltage, sub-threshold leakage is much higher. As gate oxides continue to scale, gate leakage is also becoming important. Keshavarzi, Roy, and Hawkins give an overview of these different leakage mechanisms in [20].

Our techniques for modeling gate leakage and parameter variations are described in Sections 3.2 and 3.3. Our technique for modeling sub-threshold leakage and its dependence on temperature, etc. is to extend the high-level model of sub-threshold leakage proposed by Butts and Sohi [6]. Their model neatly compartmentalizes some different issues affecting static power in a way that makes it easy to reason about leakage effects at the micro-architecture level. Leakage is given by the following equation:

$$P_{static} = V_{CC} \cdot N \cdot k_{design} \cdot \hat{I}_{leak} \quad (1)$$

This formula must be computed for each circuit or block of interest, *e.g.* the data array or a cache or the cache's "edge logic" (decoders and sense amplifiers). $V_{CC}$ is the supply voltage, and $N$ is the number of transistors in the circuit, which could be estimated by comparing it with a circuit of known functionality. $k_{design}$ is a factor determined by the specific circuit topology and accounts for effects like transistor sizing, transistor stacking and the number and relationship of NMOS and PMOS transistors in a circuit. $\hat{I}_{leak}$ is a normalized leakage value for a single transistor, which we refer to as *unit leakage*.

Using this model, it is easy to see the relationships of some major factors that a processor designer can control for leakage-power




(a) Leakage vs. Aspect Ratio (W/L)

(b) Leakage vs. Supply Voltage (Vdd)

(c) Leakage vs. Temperature (T)

(d) Leakage vs. Threshold Voltage (Vth)

**Figure 1. Comparisons of the proposed HotLeakage model against circuit-level simulation results.**

savings: given a unit leakage $\hat{I}_{leak}$, leakage power is proportional to operating voltage and the number of transistors in the unit of interest. For example, DVS affects leakage by reducing $V_{CC}$, and "turning off" some unit (a cache bank or part of an issue queue) by disconnecting its power supply effectively reduces $N$.

In the Butts and Sohi formulation, the unit leakage $\hat{I}_{leak}$ is calculated once and assumes fixed values for threshold voltage ($V_T$), operating temperature, etc. Since recent work in low-leakage cache design [11, 14, 23, 26] as well as broader processor-design issues like thermal management [4, 18, 28, 29] manipulate parameters like $V_T$ and temperature that are hidden in $k_{design}$ or $\hat{I}_{leak}$, computing one fixed value for $k_{design}$ and $\hat{I}_{leak}$ is not well-suited for actual simulation work (see [32] for more details).

To develop a portable simulation module for use with various architecture-level simulators, we retain the notions of $k_{design}$ and unit leakage but compute the unit leakage dynamically during the simulation using the BSIM3 [3] leakage-current equation. This lets us explicitly account for temperature, supply voltage, and threshold voltage as key parameters, and includes the important DIBL effect which is sensitive to supply voltage. We also use two separate $k_{design}$'s for P- and N-type.

### 3.1.1 Unit Leakage

Based on the BSIM3 v3.2 [3] equation for leakage in a MOSFET transistor, our leakage model of a single transistor is given by the following equation:

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

Low-level parameters are derived using transistor-level simulations: $u_0$ is the zero bias mobility, $C_{OX}$ is gate oxide capacitance per unit area, $W/L$ is the aspect ratio of the transistor, $e^{b(V_{dd}-V_{dd0})}$ is the DIBL factor derived from the curve fitting method, $V_{dd0}$ is the default supply voltage for each technology ($V_{dd0}$=2.0 for 180nm, $V_{dd0}$=1.5 for 130nm, $V_{dd0}$=1.2 for 100nm and $V_{dd0}$=1.0 for 70nm), $v_t = kT/q$ is the thermal voltage, $V_{th}$ is threshold voltage which is also a function of temperature, $n$ is the subthreshold swing coefficient, and $V_{off}$ is an empirically determined BSIM3 parameter which is also a function of threshold voltage. In these parameters, $u_0$, $C_{OX}$,$W/L$ and $V_{dd0}$ are statically defined parameters; the DIBL factor $b$, subthreshold swing coefficient $n$ and $V_{off}$ are derived from curve fitting based on transistor-level simulations; $V_{dd}$, $V_{th}$ and $v_t = kT/q$ are calculated dynamically in the simulations.

The above equation is based on two assumptions:

1. $V_{gs}$=0 — we only consider the leakage current when the transistor is off.
2. $V_{ds}$=$V_{dd}$ — we only consider a single transistor here; the stack effect and the interaction among multiple transistors are taken into account when we model the cell using Equation 3.

Figure 1 shows the comparison of leakage current calculated by our model to the transistor-level simulation. From Figure 1a, 1b, and 1c, we can see that for the ratio $W/L$, supply voltage $V_{dd}$ and temperature $T$, our results perfectly match the simulation results. Figure 1d shows that after threshold voltage increases to some value, the modeled leakage current does not decrease anymore. This is due to the simplicity of our model, which only considers the subthreshold leakage and DIBL effect. It is only of concern if threshold voltage is beyond the normal value.

### 3.1.2 Leakage per Cell

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 [32] 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:

$$I_{cell\_leakage} = n_n \cdot K_n \cdot I_n + n_p \cdot K_p \cdot I_p \quad (3)$$

$n_N$ and $n_P$ are the number of NMOS and PMOS transistors in the cell, and $I_N$ and $I_P$ are the calculated leakage current of NMOS and PMOS according to Equation 2; when aspect ratio $W/L = 1$ we call them *unit leakage*. $k_{design}$ is then a scaling factor determined for each type of cell to account for the transistor stack effect and the aspect ratios $(W/L)$ of the different transistors. (The stack effect refers to the additional reduction in leakage when multiple series-connected transistors are off; for example, sleep transistors take advantage of this.) This means that the expression for static power analogous to Equation 1 is:

$$P_{static} = V_{dd} \cdot N_{cells} \cdot I_{cell} \quad (4)$$

$k_n$ and $k_p$, the design factors of N and P transistors, can be derived by a similar method as in the single-$k_{design}$ model. For a given cell, we divide all possible inputs into two groups: one group inputs will turn off the pull-down network composed of N transistors. The other group will turn off the pull-up network composed of P transistors. Thus the leakage currents are also divided into two groups $I_{1n}, I_{2n},\ldots, I_{kn},\ldots$ and $I_{1p}, I_{2p},\ldots, I_{kp},\ldots$ $I_{kn}$ is the leakage current when the pull-down network is turned off, while $I_{kp}$ is the leakage current when the pull-up network is turned off. $k_n$ and $k_p$ are given by the following equation:

$$k_n = (I_{1n} + I_{2n} + \cdots + I_{kn} + \cdots)/(N * n_n * I_n) \quad (5)$$

$$k_p = (I_{1p} + I_{2p} + \cdots + I_{kp} + \cdots)/(N * n_p * I_p) \quad (6)$$

$N$ is the number of all possible combinations. For example, Figure 2 is the diagram of a two-input NAND gate. There are two

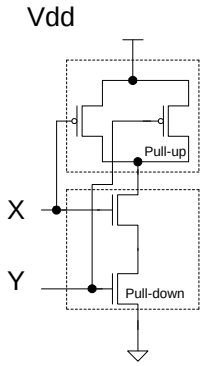



**Figure 2. Two-input NAND gate.**

inputs X and Y, which make four possible combinations. There are three combinations: $(X = 0, Y = 0)$,$(X = 0, Y = 1)$ and $(X = 1, Y = 0)$ which turn off the pull-down network. $I_{1n}$, $I_{2n}$ and $I_{3n}$ are the leakage currents corresponding to these three inputs. The only combination that turns off the pull-up network is $(X = 1, Y = 1)$ and $I_{1p}$ is the corresponding leakage current. $k_n$ and $k_p$ are given by:

$$k_n = (I_{1n} + I_{2n} + I_{3n})/(N * n_n * I_n) \quad (7)$$

$$k_p = I_{1p}/(N * n_p * I_p) \quad (8)$$

Here $N$ equals 4.

The double-$k_{design}$ model has the important property that it is able to handle the differential scaling of N and P transistors that is widely used in contemporary technologies. Again, detailed plots can be found in [32]. We find that $k_n$ and $k_p$ are independent of threshold voltage and have a linear relationship with temperature and supply voltage. We incorporate these features into our leakage model and $k_n$, $k_p$ are calculated dynamically with respect to different temperatures and supply voltages. These values are derived for different technology nodes via simulations.

### 3.2 Gate Leakage and GIDL Effect

In order to improve device performance, gate-oxide thickness is projected to scale aggressively for future technology nodes [27]. The result is that gate leakage through the gate oxide increases significantly due to the direct tunneling current. Our model includes gate leakage for 70nm technology, where gate leakage becomes dominant. To get an explicit equation for gate-leakage calculations is very difficult and also unnecessary for an architectural-level model. We use AIM-spice [1] as the circuit simulator, which includes BSIM4 among the supported models for gate leakage. Gate current parameters have been adjusted to target 40 nA/um gate leakage in 70nm technology at 1.2nm oxide thickness and 0.9 V supply voltage at room temperature (300K) as predicted in [27]. Gate leakage is strongly dependent on the gate oxide thickness $t_{ox}$ 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.

GIDL effect occurs at low gate voltage and high drain voltage bias. This effect will raise the leakage current when gate voltage goes negative. It becomes worse when biasing the substrate to negative voltage for N transistors and to positive voltage for P transistors. This will limit the reverse body-biasing (RBB) technique.

### 3.3 Parameter Variations

Device parameter variations can be divided into two categories: inter-die (die-to-die) variation and intra-die (within-die) variation.

Inter-die variation is the difference in the value of a parameter across nominally identical dies and is typically accounted as a shift in the mean of some parameter value equally across all device or structures on any one chip. For purposes of circuit design, it is usually sufficient to lump all the contributions in the inter-die variation into a single variation component with a mean and variance.

Intra-die variation is the deviation occurring spatially within any one die. It may have a variety of sources depending on the physics of the manufacturing steps. In contrast to inter-die variation (affecting all devices on any one chip equally), intra-die variation contributes to the mismatch behavior between structures on the same chip.

Due to both inter-die and intra-die parameter variations, there is significant variation in leakage power. Thus parameter variations must be taken into account in the new leakage model. Inter-die variation can be characterized as a global mean and variance while intra-die variation is more complicated. In this version our model only includes the inter-die variation.

There are four parameters which we are interested in. They are $L$: length of the transistor; $t_{ox}$: thickness of the gate oxide; $V_{dd}$: supply voltage; and $V_{th}$: threshold voltage of the transistor. For each parameter, user can give the specific mean $\mu$, variance $\sigma$, and the number of samples $N$. In the initializing phase of the simulation, $N$ gaussian distribution samples are generated and the leakage currents are also calculated accordingly. The mean of those leakage currents is used in the following simulations in order to include the effects of the parameter variations.

### 3.4 How to Use the HotLeakage Software Within an Architecture Simulator

The HotLeakage simulator is a configurable module. The various parameters related to the leakage power modeling and the leakage control techniques are specified at the command line (see [32] for details). To use HotLeakage with currents based on BSIM3 models and our pre-determined values of $k_{design}$, it is only necessary to specify the technology parameter; e.g. 70nm. Other parameters can also be configured, but all have reasonable default values.

HotLeakage dynamically tracks leakage for each cell of interest (*e.g.*, an SRAM cell) and this information is then translated into leakage at the architecture level. The functions that calculate leakage for each structure of the micro-architecture are in the main leakage module, and these need to be called whenever any of the parameters—like temperature, supply voltage, etc.—that affect leakage is changed. These functions will recalculate the leakage currents using the HotLeakage model. HotLeakage and the accompanying simulation infrastructure currently model leakage of caches and register files; adding models for other cache-like structures is very simple.

The power-performance simulator, *e.g.* Wattch, is responsible for implementing the leakage-control technique and using the HotLeakage values accordingly. As mentioned earlier, we have implemented a generic abstraction for modeling leakage control techniques based on putting individual cache lines into standby, allowing us to study techniques like gated-$V_{ss}$ [19], drowsy cache [11], and reverse-body-bias [23].

## 4 Simulation Set-Up

### 4.1 Processor Model

All simulations were performed with Wattch augmented by HotLeakage. Unless stated otherwise, this paper uses the baseline configuration as shown in Table 2, which resembles as much as possible the configuration of an Alpha 21264 [21]. The most important difference for this paper is that in the 21264 there is no separate BTB, because the I-cache has an integrated next-line predictor [9]. As most processors currently do use a separate BTB, our work models a separate, 2-way associative, 1 K-entry BTB that is accessed in parallel with the I-cache and direction predictor.

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.

For Wattch and HotLeakage technology parameters we use the process parameters 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 pessimistic.

### 4.2 Benchmarks

In our comparative evaluation of various leakage control techniques, we use 11 integer benchmarks from the SPEcpu2000 [30]

| Processor Core | |
|---|---|
| Instruction Window | 80-RUU, 40-LSQ |
| Issue width | 4 instructions per cycle |
| Functional Units | 4 IntALU,1 IntMult/Div,<br>2 FPALU,1 FPMult/Div,<br>2 mem ports |
| Memory Hierarchy | |
| L1 D-cache | Size 64 KB, 2-way LRU, 64 B blocks<br>2-cycle latency |
| L1 I-cache | Size 64 KB, 2-way LRU, 64 B blocks<br>1-cycle latency |
| L2 | Unified, 2 MB, 2-way LRU,<br>64B blocks, 11-cycle latency |
| Memory | 100 cycles |
| Branch Predictor | |
| Branch predictor | Hybrid:<br>4K bimod and 4K/12-bit/GAg<br>4K bimod-style chooser |
| Branch target buffer | 1 K-entry, 2-way |

**Table 2. Configuration of simulated processor microarchitecture. All caches are write-back.**

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.

# 5 Results

## 5.1 L2 Latency

Our results roughly duplicate those in [11]. They report slightly higher leakage savings and slightly lower performance 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 3 and 4 present the *net* cache-leakage savings and the performance loss for a system with an L2 cache latency of 5 cycles, as might be seen for a fast, on-chip L2. 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.

Figures 5 and 6 then present the same results for an 8-cycle L2; Figures 8 and 9 for an 11-cycle L2; and Figures 10 and 11 for a 17-cycle L2.

These results show that for 5–8 cycle L2 caches, gated-$V_{ss}$ is superior to drowsy cache in terms of both energy savings and performance loss. At 5 cycles, gated-$V_{ss}$ is almost uniformly superior, while at 8 cycles, drowsy is superior for a small number of benchmarks. At 11 cycles, the picture is less clear. Gated-$V_{ss}$ is slightly better in terms of average energy savings and slightly worse in terms of average performance loss. But looking at individual benchmarks, drowsy and gated-$V_{ss}$ are better for about an equal number of benchmarks. Finally, at 17 cycles, drowsy cache becomes clearly superior.

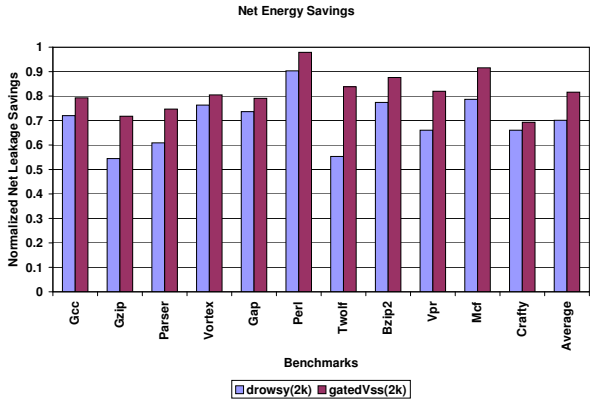



**Figure 3. Net leakage savings at 110° and an L2 latency of 5 cycles.**

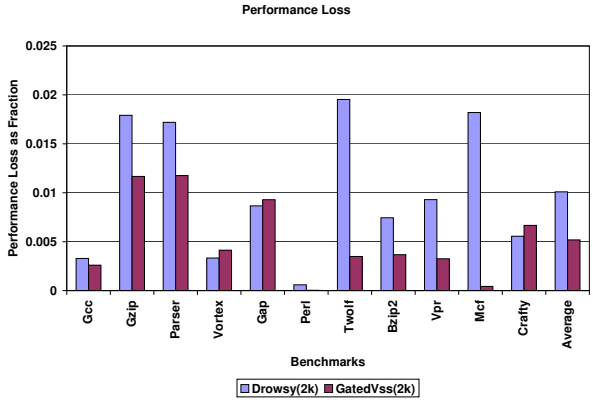



**Figure 4. Performance loss at an L2 latency of 5 cycles.**

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

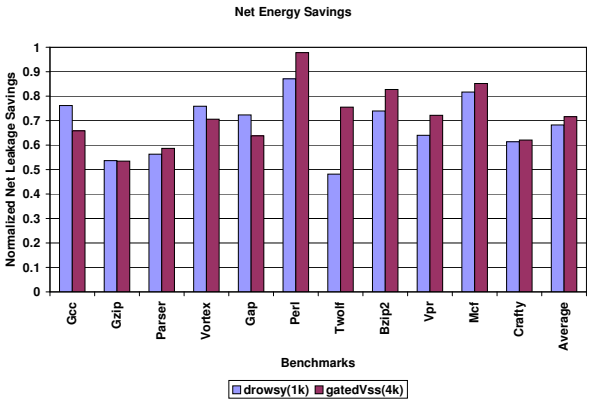



**Figure 5. Net leakage savings at 110° and an L2 latency of 8 cycles.**

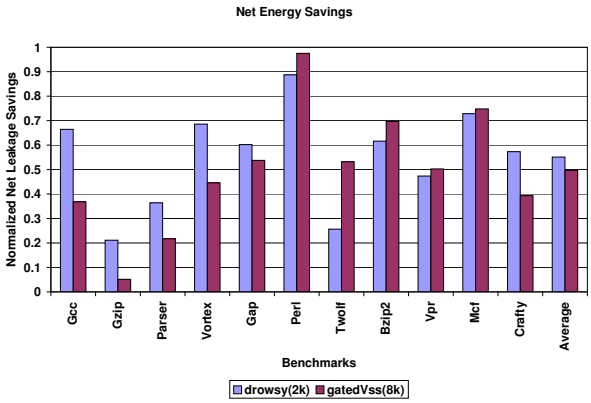



**Figure 7. Net leakage savings at 85° and an L2 latency of 11 cycles.**

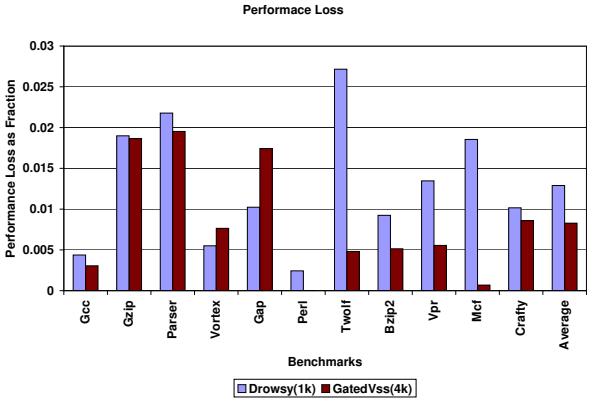



**Figure 6. Performance loss at an L2 latency of 8 cycles.**

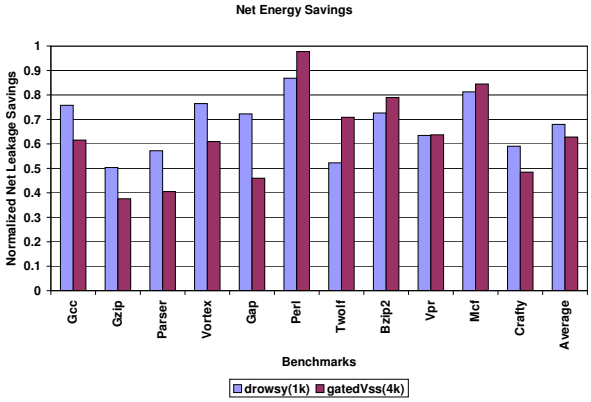



**Figure 8. Net leakage savings at 110° and an L2 latency of 11 cycles.**

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-$V_{ss}$ was no longer able to hide a significant portion of L1 miss times, and the state-preserving nature of drowsy cache becomes a major advantage.

## 5.2 Temperature

Figures 7 and 8 illustrate the effects of temperature for an 11-cycle L2 cache by comparing energy savings at 85°C and 110°C. Because leakage is exponentially dependent on temperature, the energy savings is much higher for both schemes.

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

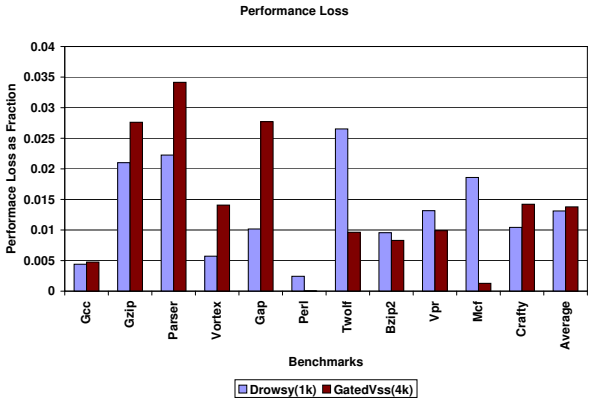



**Figure 9. Performance loss at an L2 latency of 11 cycles.**

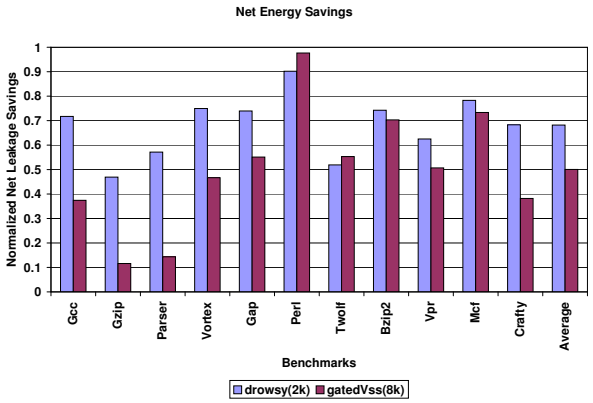



**Figure 10. Net leakage savings at 110° and an L2 latency of 17 cycles.**

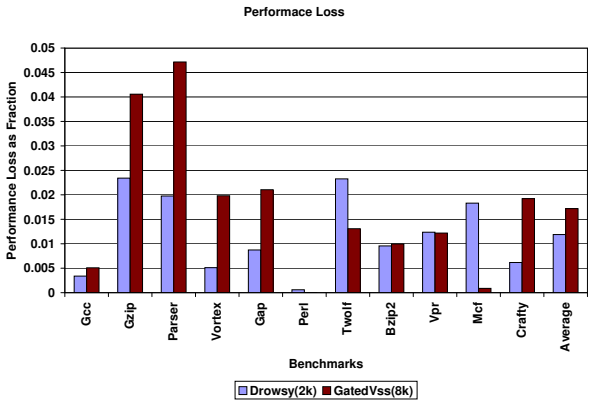



**Figure 11. Performance loss at an L2 latency of 17 cycles.**

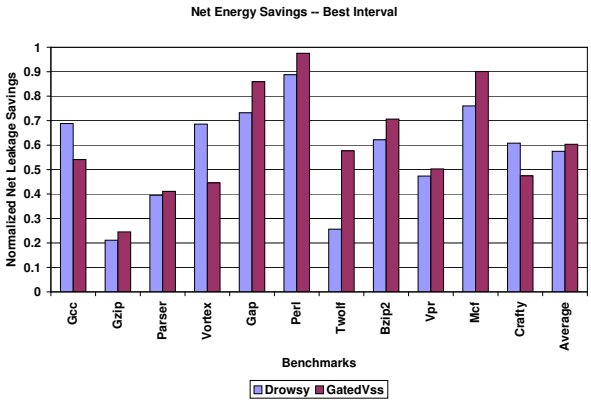



**Figure 12. Net leakage savings at 85° and an L2 latency of 11 cycles for the best per-benchmark decay interval.**

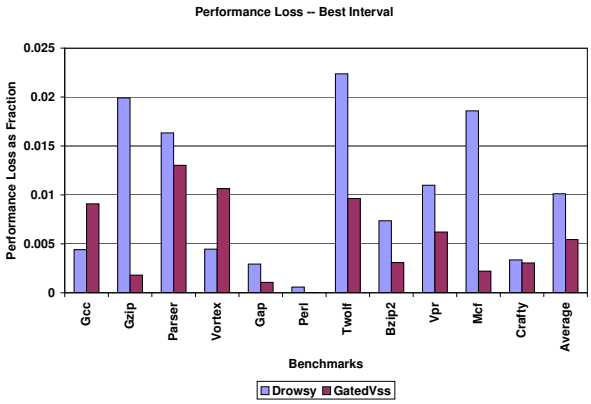



**Figure 13. Performance loss at an L2 latency of 11 cycles for the best per-benchmark decay interval.**

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 little impact on the relative performance of gated-$V_{ss}$ and drowsy.

## 5.3 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.

## 5.4 Adaptivity

Figures 12 and 13 show 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-$V_{ss}$, we identify the best decay interval for each benchmark, and these are the results that are plotted. The best intervals are itemized in Table 3.

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 ILP that can be used to hide induced misses. Comparing Figures 12 and 13 against Figures 7 and 9 shows that using the best per-benchmark intervals improves energy savings for gated-$V_{ss}$ by 20%, from 50% to 60%, and dramatically reduces performance loss, from about 1.4% to about 0.55%. Energy savings for drowsy cache only improve by about 4% and performance loss only improves from 1.3% to 1.0%.

It is to be expected from the analysis in [11] that adaptivity is not necessary for drowsy cache, because for reasonable intervals, it is fairly insensitive to decay interval. Gated-$V_{ss}$ does not need

| | Drowsy | Gated-$V_{ss}$ |
|---|---|---|
| Gcc | 1k | 2k |
| Gzip | 2k | 64k |
| Parser | 4k | 16k |
| Vortex | 2k | 8k |
| Gap | 16k | 16k |
| Perl | 4k | 4k |
| Twolf | 2k | 4k |
| Bzip2 | 4k | 16k |
| Vpr | 2k | 8k |
| Mcf | 1k | 2k |
| Crafty | 4k | 32k |

**Table 3. Best decay intervals.**

adaptivity to give attractive benefits for on-chip L2 caches, but performs much better with adaptive decay intervals. It becomes clearly superior to drowsy for most benchmarks with an 11-cycle L2.

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. [19]; the *adaptive mode control* technique proposed by Zhou et al. [33]; and the formal feedback-control technique proposed in our prior work [31]. The latter two techniques require the tags to stay awake. Our feedback-control technique is quite simple, using the tags to identify induced misses and requiring only a small state machine to periodically update the counter containing the decay interval.

## 6 Conclusions and Future Work

HotLeakage provides the first publicly-available microarchitecture-level leakage-modeling 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 models) for modeling cache and register files, and provides a simple interface for selecting alternate parameter values and for modeling 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 at `http://lava.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. 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.

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

[2] A. Alvandpour, R. Krishnamurthy, K. Soumyanath, and S. Borkar. A low-leakage dynamic multi-ported register file in 0.13um CMOS. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 68–71, Aug. 2001.

[3] U.C. Berkeley. BSIM3 v3.1 SPICE MOS device models, 1997. http://www-device.EECS.Berkeley.EDU/~bsim3/.

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 171–82, Jan. 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[6] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 191–201, Dec. 2000.

[7] A. Buyuktosunoglu, D. H. Albonesi, P. Bose, P. W. Cook, , and S. E. Schuster. Tradeoffs in power-efficient issue queue design. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, Aug. 2002.

[8] A. Buyuktosunoglu, S. E. Schuster, D. Brooks, P. Bose, P. W. Cook, and D. H. Albonesi. An adaptive issue queue for reduced power at high performance. In *Workshop on Power-Aware Computer Systems*, Nov. 2000.

[9] B. Calder and D. Grunwald. Next cache line and set prediction. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 287–96, June 1995.

[10] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing static leakage energy in microprocessor functional units. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–32, Nov. 2002.

[11] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 147–57, May 2002.

[12] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 248–59, June. 2001.

[13] H. Hanson. Personal communication, May 2003.

[14] H. Hanson et al. Static energy reduction techniques for microprocessor caches. In *Proceedings of the 2001 International Conference on Computer Design*, pages 276–83, Sept. 2001.

[15] S. Heo, K. Barr, M. Hampton, and K. Asanović. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 137–47, May 2002.

[16] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D. W. Clark. Managing leakage for transient data: Decay and quasi-static memory cells. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 52–55, Aug. 2002.

[17] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi. Applying decay strategies to branch predictors for leakage energy savings. In *Proceedings of the 2002 International Conference on Computer Design*, pages 442–45, Sept. 2002.

[18] W. Huang, J. Renau, S.-M. Yoo, and J. Torellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 202–13, Dec. 2000.

[19] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[20] A. Keshavarzi, K. Roy, and C. F. Hawkins. Intrinsic leakage in low power deep submicron CMOS ICs. In *Proc. of the 1997 International Test Conference*, pages 146–55, Nov. 1997.

[21] R. E. Kessler, E. J. McLellan, and D. A. Webb. The Alpha 21264 microprocessor architecture. In *Proceedings of the 1998 International Conference on Computer Design*, pages 90–95, Oct. 1998.

[22] S. R. Nassif. Modeling and forecasting of manufacturing variations. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, 2001.

[23] K. Nii et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 293–98, Aug. 1998.

[24] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 248–59, Dec. 2001.

[25] 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 *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 90–95, July 2000.

[26] K. Roy. Leakage power reduction in low-voltage CMOS designs. In *Proceedings of the International Conference on Electronics, Circuits, and Systems*, pages 167–73, 1998.

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

[28] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 17–28, Feb. 2002.

[29] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, Apr. 2003.

[30] Standard Performance Evaluation Corporation. SPEC CPU-2000 Benchmarks. http://www.specbench.org/osg/cpu2000.

[31] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*, May 2002.

[32] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia Department of Computer Science, Mar. 2003.

[33] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.