Circuit and Microarchitectural Techniques for Processor On-Chip Cache Leakage Power Reduction

by

Nam Sung Kim

A dissertation proposal in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2004

To my family, with love and thanks

Acknowledgements

Many people contributed to the success of this work and while I would like to acknowledge individually each by name, I would inevitably leave out deserving friends and relatives. Even the short list contained in these paragraphs is likely incomplete. I apologize in advance for such omission and convey my deepened respect and admiration to all who contributed to the extraordinary experiences I have been fortunate to enjoy.

Foremost I thank my family who has been a tremendous source of love, encouragement, and inspiration. The support from my parents Kyung-Jung Kim and Young-Soon Choi, my lovely wife Seong Hye Hwang, my sister Hee-Sun Kim, and my brother Nam-Gu Kim kept me going not only through this specific task but through my entire life. Especially, I'd like to thank my wife, Seong Hye for her endless love and encouragement. Without her, I might not be able to finish my study.

Beyond "family" support, Trevor Mudge, my advisor, has certainly been a major supporter over the past four years. He has taken care of me like his own son and tried to keep encouraging me whenever I am depressed or disappointed by events. I was very lucky to be his student as soon as I came to the University of Michigan and I owed him too for things including this dissertation and the research papers we wrote together. I also wish to thank the entire dissertation committee members, David Blaauw, Todd Austin, Steve Reinhardt, and Dennis Sylvester, for their insight and guidance.

In addition, I thank David Green, Rajeev Krishna, and Nathan Binkert who did great work in maintaining our simulation pool system and Taejoon Park, Joohee Kim, Songkuk Kim, Gijoon Nam, and Taeho Kgil who gave me much valuable advice on various topics. I also thank Eric Larson who kindly answered my questions about hacking and modifying the SimpleScalar and MASE simulators.

Finally, I would also like to thank those who provided the financial backing for my graduate studies. I have been supported by DARPA contract number F3361500-C (September 2000 to August 2001), SRC 2001-HJ-904 (September 2001 to August 2002), and an Intel Graduate Fellowship (September 2002 to August 2003). Simulations were performed on computers donated through the Intel Education Grant.

Table of Contents

List of Figures

List of Tables

List of Appendices

Chapter 1
Introduction

1.1 Motivation

Power consumption is now the major technical problem facing the semiconductor industry. In comments on this problem at the 2002 International Electron Devices Meeting, Intel chairman Andrew Grove cited off-state current leakage in particular as a limiting factor in future microprocessor integration [1]. Off-state leakage is *static power*, current that leaks through transistors even when they are turned off. It is one of two principal sources of power dissipation in today's microprocessors. The other is *dynamic power*, which arises from the repeated charge and discharge of the capacitance on the output of the hundreds of millions of gates in today's chips.

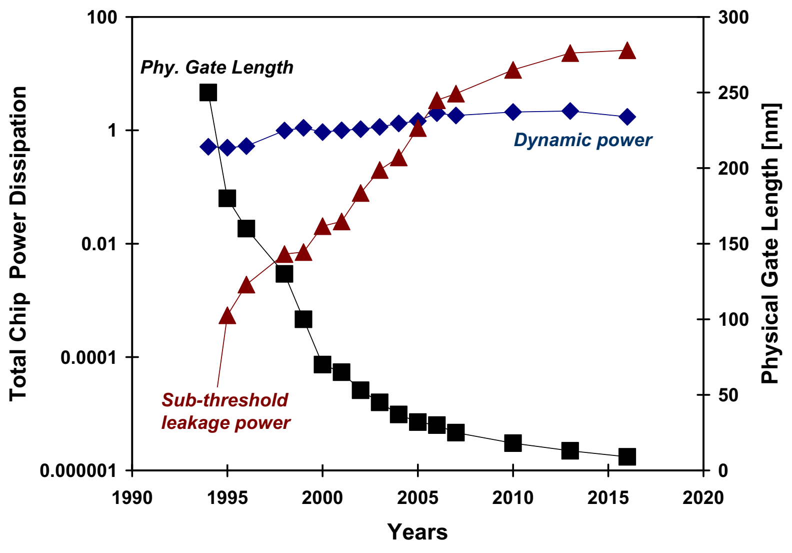
Until very recently, only dynamic power has been a significant source of power consumption, and Moore's law has helped to control it. Shrinking processor technology has allowed and, below 100 nanometers, actually required reducing the supply voltage, so reducing the voltage significantly reduces power consumption. Unfortunately, smaller geometries exacerbate leakage, so static power begins to dominate the power consumption equation in microprocessor design.

1.2 Process Trends

Historically, *complementary metal-oxide semiconductor* (CMOS) technology has dissipated mush less power than earlier technologies such as *transistor-transistor logic* (TTL) and *emitter-coupled logic* (ECL). In fact, when not switching, CMOS transistors lost negligible power. However, the power they consume has increased dramatically with increases in device speed and chip density. The research community has recognized the significance of this increase for some time. Figure 1.1 also shows exponential increase projected for a principal component of static power consumption — sub-threshold leakage, a weak inversion current across the device.

The *international technology roadmap for semiconductor* (ITRS) expects the rate of this increase to level out in 2005 but to remain substantial nonetheless. In a few years, total power dissipation from chip static power will exceed the total from dynamic power,

Figure 1.1: Total chip dynamic and leakage power dissipation trends.



Two power plots for sub-threshold leakage and dynamic power represent the 2002 projections normalized to those for 2001 published by international technology roadmap for semiconductor (ITRS) [2]. The power increases assume a doubling of on-chip devices every two years in line with Moore's law.

and the projected increases in off-state sub-threshold leakage show that it will exceed total dynamic power consumption as technology drops below the 65*nm* feature size [3]. As leakage current becomes the major contributor to total chip power consumption, the industry must reconsider the power equation that limits system performance, chip size, and cost.

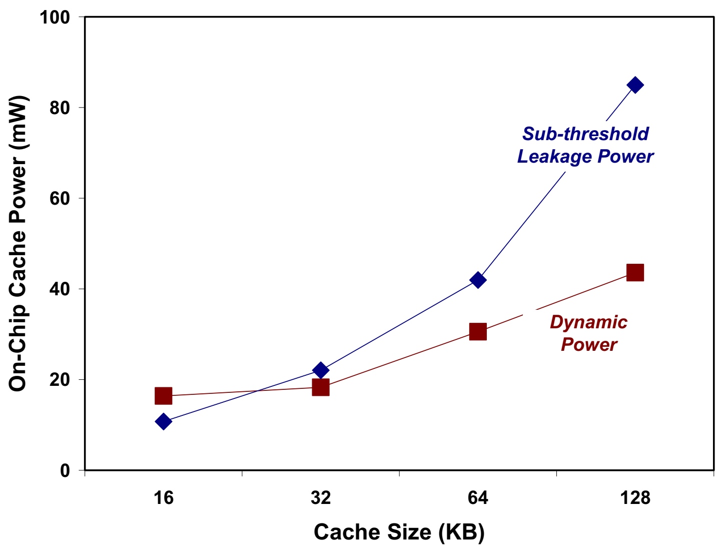
1.3 On-Chip Cache Leakage

The sub-threshold leakage power is now one of the most difficult issues confronting microprocessor designers. On one hand, performance demands require the use of fast transistors that consume leakage power even when they are turned off. On the other hand, new applications and cost issues favor designs that are energy efficient. Leakage power is a problem for all microprocessor circuit components, but it is a particularly important problem in processor on-chip caches where a large number of potentially high-leakage cross-coupled inverters — the storage elements of caches — are integrated in great numbers.

Moreover, the cache leakage power will be the dominant fraction of total processor power dissipation, because larger L1, L2, and even L3 on-chip caches are being integrated on the die. For example, Intel’s Madison processor has 1MB and 6MB on-chip L2 and L3 caches respectively, and the leakage power will be the dominant fraction of total power consumption of those caches.

Figure 1.2 shows the dynamic and leakage power trends for 16KB, 32KB, 64KB, and 128KB direct-mapped caches designed with a 70*nm Berkeley predictive technology model* (BPTM) [4] and sub-banking techniques [5]. As the size of the caches increases, the fraction of the leakage power exceeds that of the dynamic power. While dynamic power is

Figure 1.2: On-chip cache dynamic and leakage power trends.



On-chip caches are designed with the 70nm Berkeley predictive technology model (BPTM) [4] and sub-banking techniques. 4KB sub-banks are used for the 16KB and 32KB caches and 8KB sub-banks are used for 64KB and 128KB caches, respectively.

consumed only when those caches are accessed, static leakage power is always dissipated even though they are not accessed. Therefore, the upper-level (higher number) caches such as L2 or L3 more severely suffer from leakage power loss because they are only accessed by lower-level cache misses. For instance, according to our experiment with the projected 70*nm* CMOS process [4], leakage power could consume as much as 87% of the power in 1MB caches if left unchecked [6].

1.4 Thesis Contributions

In this thesis, we propose simple but effective circuit and microarchitectural control techniques to reduce the leakage power of caches while minimizing any attendant performance loss. The emphasis and novelty of these proposed techniques are simple circuits and simple microarchitectural mechanisms to control those circuits effectively. There is a synergy: the proposed circuits help the computer architects simplify the microarchitectural controls. The emphasis on simplicity is also a key point. Complex microarchitectural techniques are unattractive because they add to validation time, and ultimately time-to-market.

1.4.1 A Low-Leakage, State-Preserving 6-Transistor Memory Cell

To reduce the leakage power dissipation of memory elements, we propose applying dynamic voltage scaling to 6-transistor memory cells. While the previously proposed circuit techniques either destroy the memory cell states or have slow *wake-up* from stand-by to active mode, our proposal is able to preserve the memory cell states with a fast wake-up while reducing the leakage power noticeably. These circuit properties also simplify the microarchitectural controls needed for wake-up.

In this thesis, we also provide detailed circuit simulation results analyzing the leakage power reduction as well as characterizing the robustness of the proposed circuit in the presence of possible noise sources.

1.4.2 Microarchitectural Controls for Low-Leakage Caches

First, we characterize the working sets[1] of on-chip L1 caches to investigate effective microarchitectural controls for the leakage power reduction. According to our analyses, the working sets of both instruction and data caches are quite small. This means that the rest of the cache may be put into stand-by mode to reduce the leakage power without significant performance losses. However, the working sets of the data caches change significantly in a few thousand-cycle periods while those of instruction caches do not.

1. A set of cache lines that have been accessed at least once during a specified number of cycles.

Second, we introduce microarchitectural mechanisms specialized for *data* and *instruction* caches to balance leakage power reduction and performance impact. They are suggested by our working-set analyses. The two types of caches have very different access patterns. Therefore, they require a specific control mechanism for each type of cache. Both techniques take full advantage of the proposed circuit technique.

Finally, the instruction cache access location patterns obtained from our working-set analyses are used to support a proposed *gated bit-line precharge* technique to further reduce the leakage power from the supply voltage source to the bit-lines, by gating the precharge clock signal. In this technique, we precharge only a sub-set of cache sub-banks. We reduce the performance loss that occurs from the cycles taken to initiate a precharge to a newly accessed sub-bank by precharging the sub-bank predictively. The predictor used for this technique has only small number of entries, but shows very high accuracies for most workloads in the entire SPEC2K suite.

1.4.3 Leakage Optimization via Multiple-Threshold Voltage Assignments

First, assuming that there are multiple threshold voltages (V_{TH}'s) available, we partition a cache into four components, and we present the leakage power and access time models for each component parameterized by size. Based on these models, we propose optimization techniques to reduce the leakage power of individual on-chip caches. The objective of the optimization is to find a set of V_{TH}'s that minimizes the cache leakage power for a given access time constraint.

Second, we investigate leakage power reduction and access time increase trends when increasing the V_{TH}'s of the components. We also present V_{TH} trends of each cache component during the leakage power optimization for various numbers of V_{TH}. These trends will play a key role in guiding future research on the design of low-leakage upper-level cache circuits.

Finally, we extend the techniques to the leakage power reduction of multi-level caches under the constraint of maintaining the same average cache memory system performance. In this investigation, we exploit cache *hit* and *miss* characteristics to optimize the leakage power of multi-level caches. In contrast, previous work on the cache leakage power reduction has been focused only on either circuit design or microarchitectural control aspects.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces the proposed circuit and microarchitectural techniques to reduce the leakage power of *static random access memory* (SRAM) — a major component of caches. Chapter 3 proposes a low-leakage, state-preserving 6-transistor memory cell circuit using a dynamic voltage scaling technique. Chapter 4 investigates data cache working set characteristics, and examines various microarchitectural controls to minimize the cache leakage power and run-time increase. Chapter 5 shows that the microarchitectural control mechanism used for data caches does not work well for instructions caches due to different working set characteristics or access patterns, and proposes a more sophisticated control for instruction caches. Chapter 6 presents cache leakage optimization techniques based on the simultaneous assignments of multiple-threshold voltages. It evaluates access time impact and leakage power reduction of the resulting memory system. Chapter 7 contains concluding remarks and suggests future research directions. The appendices provide detailed simulation parameters, models, and results.

Chapter 2
Background

This chapter on background work briefly outlines the various research areas that are relevant work to reducing the sub-threshold leakage power dissipation of *static random access memory* (SRAM). We start by discussing the circuit and device techniques. We then look at microarchitectural and compiler techniques that combine with the circuit and device techniques.

2.1 Circuit and Device Techniques

2.1.1 Multi-Threshold CMOS SRAM

The *multi-threshold CMOS* (MTCMOS) circuit technique was proposed to satisfy both the requirement of lowering the threshold voltage of transistors and reducing stand-by sub-threshold leakage current [7]. To increase the circuit speed, *low*-V_{TH} transistors are used for logic gates and during long stand-by times (i.e., sleep time), the power supply is disconnected with *high*-V_{TH} transistors. This concept was also applied in the proto-design of MTCMOS memories to reduce the power dissipation of peripheral circuitry as shown in Figure 2.1 [8, 9].

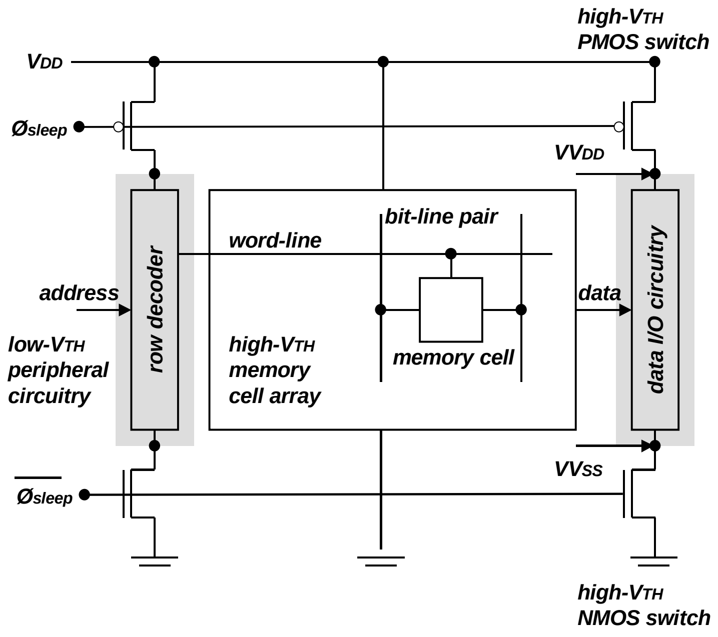


Figure 2.1: Multi-Threshold CMOS SRAM Architecture.

In Figure 2.1, the row decoder and data I/O circuitry that require a very high circuit speed consist of low-V_{TH} transistors. Their power terminals are not connected directly to the power supply lines — V_{DD} and V_{SS}, but rather to *virtual* power supply lines — VV_{DD} and VV_{DD}. The high V_{TH} transistors serve as sleep control transistors and link the real and virtual power lines.

The MTCMOS technique cannot be applied to the memory cells because the sleep control transistors cut the power supply off, which destroys the memory cell states. However, the memory cells are responsible for the majority of leakage power dissipation in an SRAM. To cut off the leakage power dissipation of memory cell arrays, high-V_{TH} transistors are used as illustrated in Figure 2.1, but this increases the overall memory access time. Furthermore, MTCMOS does not work below 0.6*V*, because the high-V_{TH} transistors will not turn on. Therefore, the MTCMOS cannot be used in sub-1*V* applications.

2.1.2 Adaptive Body-Biasing CMOS SRAM

Previously, the *adaptive reverse body-biasing* or *variable threshold* CMOS techniques have been proposed to control the leakage current during stand-by mode [10, 11, 12, 13]. Also, methods using *forward body-biasing* have also been proposed [14, 15, 16]. Recently, a simultaneous application of both the reverse and forward body-biasing technique has been proposed [17].

In general, the sub-threshold leakage current decreases exponentially as V_{TH} increases, and the adaptive body-biasing technique has the advantage that it reduces the leakage current exponentially by increasing V_{TH}. The V_{TH} of a short-channel MOSFET transistor in the BSIM [18] model — a physics-based, accurate, scalable, and predictive MOSFET SPICE model for circuit simulation and CMOS technology development — is given by:

$$V_{TH} = V_{TH0} + \gamma(\sqrt{\Phi_s - V_{bs}} - \sqrt{\Phi_s}) - \theta_{DIBL} V_{DD} + \Delta V_{NW} \qquad \text{(Eq. 2-1)}$$

where V_{TH0} is the zero-bias threshold voltage, Φ_s, γ, and θ_{DIBL} are constants for a given technology, V_{bs} is the voltage applied between the body and source of the transistor, ΔV_{NW} is a constant that models narrow width effects, and V_{DD} is the supply voltage [18, 19]. To increase the V_{TH}, ABB MTCMOS techniques decrease V_{bs} adaptively.

Figure 2.2 shows the adaptive reverse body-biasing MTCMOS SRAM circuit proposed in [20]. There are four additional high-V_{TH} transistors — Q_1, Q_2, Q_3, and Q_4. In the active mode, $\phi_{sleep} = V_{SS}$ is applied and Q_1, Q_2, and Q_3 are turned on and Q_4 is turned off. Then both VV_{DD} and the substrate bias, *BP*, becomes 1.0*V*. On the other hand, in the sleep

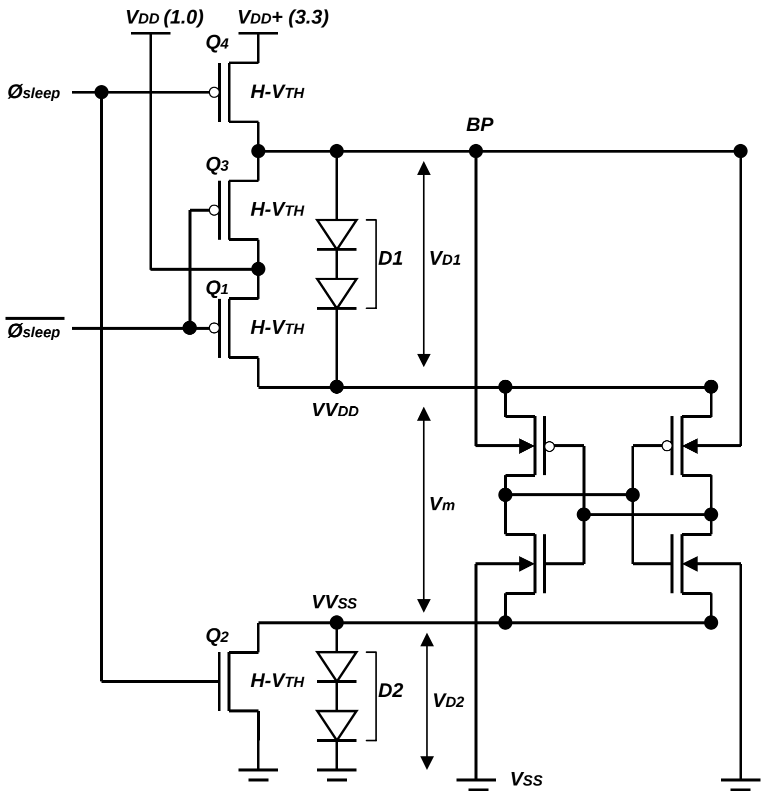


Figure 2.2: ABB-MTCMOS SRAM Circuit.

mode, ϕ_{sleep} = V_{DD} is applied and Q_1, Q_2, and Q_3 are turned off and Q_4 is turned on. In this case, *BP* becomes 3.3*V*, which decreases V_{bs} resulting in increasing the V_{TH} of PMOS transistors in the memory cells. The sub-threshold leakage current, which flows from V_{DD} to V_{SS} through D_1 and D_2, determines the voltages V_{D1}, V_{D2}, and V_m. Here V_{D1} denotes the bias between the source and substrate of the PMOS transistors, *VD2* denotes that of NMOS transistors, and V_m denotes the voltage between V_{DD} and V_{SS}. Although the memory architecture using the ABB MTCMOS technique reduces a substantial amount of the leakage power, the transition between the active and sleep modes is very slow because Q_2 and Q_4 have to drive large substrate capacitances.

As a variant of [20], a technique applying body-biasing to the NMOS transistors of the memory cells and the access transistor was proposed in [21]. In this technique, an RC decay circuit controls the body-biasing voltage. If memory cells connected to a word-line are not accessed for a specified time window, the charge stored in the RC circuit is completely discharged. As soon as the RC circuit is discharged below a certain specified level, the body-biasing circuit causes a switch to stand-by mode. Whenever the word-line is accessed, the charge in RC decay circuit is restored immediately.

The reverse body-biasing technique can reduce the sub-threshold leakage power via body effect. It avoids affecting the access time by switching to zero body-biasing in active mode. However, large latency and energy overheads are imposed by the transition of the body-bias circuit caused by the body V_{bs} swing and substrate capacitance. Furthermore, this technique becomes less attractive in scaled technologies because the body coefficient γ decreases with smaller dimension, and *band-to-band tunneling* (BTBT) becomes enhanced by the reverse body biasing.

As an alternative solution for this problem, an adaptive forward body-biasing MTCMOS SRAM was proposed in [22]. Basically, the forward body-biasing technique uses super high-V_{TH} transistors engineered by super-halo 2-D doping. When the transistors are in active mode, the forward body-biasing is applied to active SRAM cells. This lowers the V_{TH} of the transistors to enhance the circuit speed of the active SRAM cells while the high-V_{TH} transistors suppress sub-threshold leakage of the rest of the stand-by SRAM cells.

All adaptive body-biasing techniques impose a large voltage swing on the highly capacitive transistor substrate or body during switching between active and stand-by modes. This incurs a large dissipation of energy as well as a time penalty during the mode transitions.

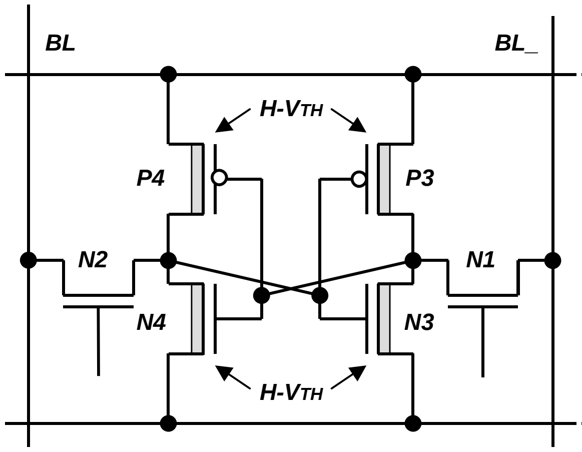
2.1.3 Dual-Threshold Voltage CMOS SRAM

In order to reduce the sub-threshold leakage current of digital circuits, dual-threshold voltage CMOS techniques have been investigated [23, 24, 25, 26]. The basic concept of using dual threshold voltage is to use low-V_{TH} transistors for circuits in the critical path, and high-V_{TH} ones for the rest of the circuits to suppress unnecessary leakage current.

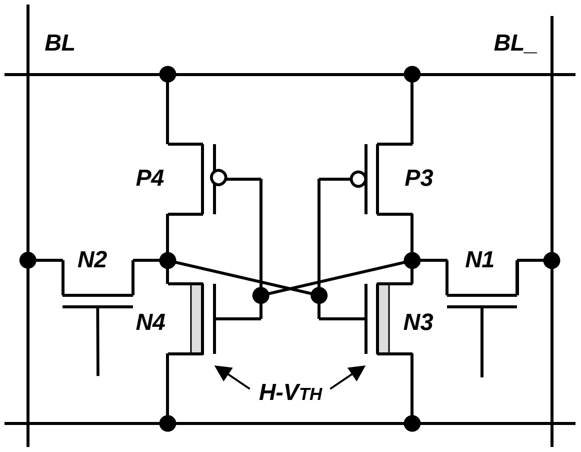
In dual-V_{TH} SRAM design, the low-V_{TH} transistors have been used in the peripheral circuits of the caches and the high-V_{TH} for the memory cells [27]. A dual-V_{TH} cell, with a high V_{TH} for the memory cell core and a low V_{TH} for both the bit lines and word lines with under-drive, has also been evaluated for the caches with differential low-swing sensing in a sub-1V V_{DD} regime [28]. However, neither of these techniques can improve the bit line delay in high-performance processor designs, because they use a single maximum V_{DD} dictated by gate-oxide wear-out considerations.

In [29, 30], different dual-V_{TH} cells and cache design choices were investigated for high performance processors with a single V_{DD} in 130*nm* technology. Figure 2.3 shows two different SRAM cell design using the dual-V_{TH} technique. As another design choice, high-V_{TH} transistors can be used for all the 6 transistors in the memory cell. However, using a cell entirely consisting of high-V_{TH} increases the delay of the cache memory up to 26% compared to that of the memory designed with the low-V_{TH} transistors according to the experimental results presented in [29, 30].

Figure 2.3: Different 6T SRAM cell designs using the dual-V_{TH} technique.



(a) DVTC-I.



(b) DVTC-II.

2.1.4 Gated-Ground CMOS SRAM

The *gated-ground* or *gated-V_{DD}* structure was introduced in [31]. This technique reduces the leakage power by using a high-V_{TH} transistor between VV_{SS} and V_{SS} to turn off the power of the memory cell when the cell is set to low-power mode as shown in Figure 2.4. This high-V_{TH} gating transistor significantly reduces the leakage power of the memory cell circuit because of the stacking effect [32, 33, 34, 35] and the exponential dependence of the leakage on V_{TH}. It uses 0.2V and 0.4V for low- and high-V_{TH} transistors, respectively and reduces the leakage power by 97% while increasing read time by 8% with 5% area overhead.

While this method is very effective at suppressing leakage, its main disadvantage lies in that it loses any information stored in the memory cell when switched into low-leakage mode. This means that a significant performance penalty may be incurred when the data in the memory cells are needed — they must be obtained from higher level caches

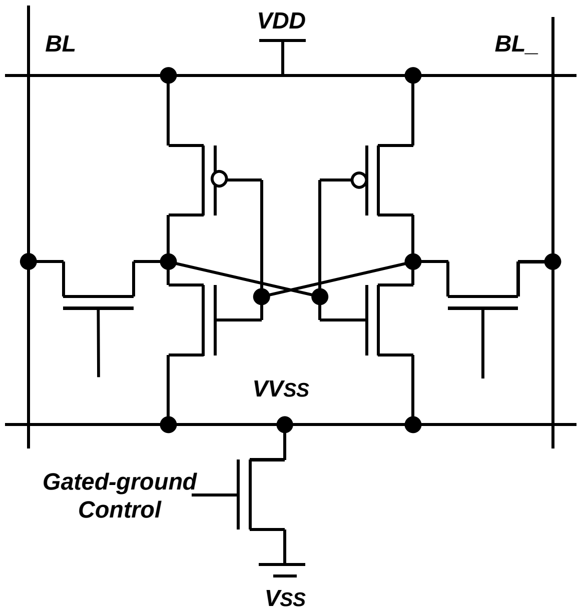


Figure 2.4: The gated-ground SRAM circuit.

or memory. Furthermore, the stacked transistor to reduce the leakage current is in the critical path. This results in increased cache access time.

In [36, 37], a single-V_{TH} *data-retention gated-ground* (DRG) SRAM was proposed. This technique relies solely on the forced-stacking effect to reduce the leakage current. To retain the cell state, sophisticated transistor sizing is required, which is sensitive to noise during sleep mode. This technique reduces leakage power by 40% while increasing read access time by 4.4% compared to a conventional memory cell designed with $0.25V$ V_{TH}. The leakage reduction by this technique is relatively lower than that of the previous technique that does not provide data-retention capability.

2.1.5 Other CMOS SRAM Techniques

Recently, a couple of other CMOS SRAM cell design techniques for high-performance processor caches were proposed to reduce leakage power [38, 39, 40]. All these circuit techniques rely, for their effectiveness, on the behavior of the microarchitecture. For example, the bits in the on-chip L1 data caches are highly biased to state “0” [38, 39], or the bits in the *branch predictor* or *branch target buffer* (BTB) are quite *transient* and *predictive* [40]. The term “transient” means that data that has not been accessed for a sufficiently long time is no longer useful (“decayed” or “dead”), and the term “predictive” implies that allowing a value to leak away, even if it will be used again, does not harm the correctness of the execution of the program. Using a decayed value will possibly cause a misprediction, but that can be corrected by existing hardware. This is a key difference compared to caches, where using decayed data will lead to incorrect execution of the program.

In [38], high-V_{TH} transistors were asymmetrically used in the SRAM cells for the selected biased state — “1” or “0.” To compensate for the slow access time of the biased state, a special sense-amplifier circuit was also proposed. This requires two more transistors per sense-amplifier. To suppress the leakage current from the supply voltage source to the bit-lines and the access transistors of the SRAM cell arrays, a leakage-biased bit-line architecture was proposed with the dual-V_{TH} storage cells in [39]. In this technique, gating the precharge devices floats the bit-lines in inactive sub-banks. This makes the leakage currents from the bit cells automatically bias the bit-line to a mid-rail voltage that minimizes the bit-line leakage current through the access transistors.

The leakage reduction of this technique depends on the percentage of zero or one resident bits in caches. However, the floated sub-bank, by gating the precharge circuits, requires a finite bit-line recharge time to initiate a new bit-line precharge. This incurs a processor performance penalty that is around 2.5% according to the reported result.

In [40], a quasi-static 4-transistor SRAM cell was proposed for the transient and predictive bits in the memory structure used for the branch prediction. The 4-transistor cells are about as fast as 6T cells, but they do not store charge indefinitely due to leakage. Thus, it can only be applied to memories used for prediction that do not affect the correctness of program execution.

2.2 Microarchitectural Techniques

Most microarchitectural and compiler leakage suppression techniques work together with the circuit techniques presented in Section 2.1. In [31, 41, 42, 43], dynamically re-sizable instruction cache architectures were proposed. The key observation behind these techniques is that there is a large variability in instruction cache utilization both within and across programs leading to large energy inefficiency in conventional caches. While the memory cells in the unused section of caches are not actively referenced, they leak current and dissipate energy.

One approach that uses the fact that much of the cache is not actively referenced is the idea of a dynamically resizing cache. This approach resizes caches to increase or decrease the number of sets in use in the caches by turning them on and off using the circuit technique proposed in Section 2.1.4. However, in [31, 41, 42, 43], the states of the cache cells are lost when the gated-ground transistor is turned off. The techniques require extra hardware for estimating cache miss rate and the cache resizing factor. The dynamically resizing cache also requires tag bits to be compatible with both small and large caches. Resizing requires special hardware and extra overhead for controlling the gated-ground transistor.

Finally, there are two sources of increase in the cache miss rate when resizing. First, resizing may demand re-mapping of data into the caches and incur a large number of compulsory misses at the beginning of the sense interval[1]. The resizing overhead depends on both resizing frequency and sense-interval length. Second, downsizing may be sub-optimal and result in a significant increase in the miss rate, when the required cache size is slightly below a given size. The impact on the miss rate is highest at small size caches when the cache begins to thrash. It was also reported that it might not be possible to downsize the cache on the fly for some applications, because they experience a large performance penalty [36, 37]. The leakage power for such application may not be reduced. Also these techniques are only applicable to L1 and are very difficult to implemented for other levels of hierarchy such as L2 or L3.

The approaches proposed in [44, 45, 46, 47] exploit the generational behavior—temporal locality—of data caches to reduce the leakage power. It turns off a "dead" cache line if a preset number of cycles have elapsed since it was last accessed. To turn off cache lines selectively, it also uses the same circuit technique proposed in Section 2.1.4. This technique incurs a cache miss penalty when the turned-off cache lines are required again. More sophisticated adaptive techniques to determine the dead cache lines were also proposed to reduce this penalty.

To reduce the leakage power dissipation through the cache bit-lines, two microarchitectural prediction techniques were proposed [48, 49] based on the circuit technique in [39]. In those techniques, they enable a sub-set of cache sub-arrays or sub-banks by gating bit-line precharge signals selectively. Because the precharge of bit-lines is in the critical

1. The sense interval is the time between consecutive cache resizes.

path of cache access time, the microarchitectural control mechanism must enable a next target or to-be-accessed sub-array before it is accessed if any penalty is to be reduced. If it wakes up a wrong sub-array, the processor must wait until the sub-array is fully precharge. This usually incurs a one cycle pseudo cache miss-penalty.

For instruction caches, they exploited both the *temporal* and *spatial locality* that arises when the program counter remains in a specific code region for a while. In [48], a *most-recently-used* (MRU) technique is used for predicting the next sub-array to precharge. The most-recently accessed sub-array index is predicted as the next target sub-array. In [49], a decay counter per sub-array is implemented to capture the recent usage of the sub-array. The value of the counter is compared to a threshold value every cycle to determine whether the sub-array is *hot* or *cold*. If the counter value is below the threshold, the sub-array was accessed recently and is likely to be reused soon. Whenever the sub-array is accessed, the decay counter is reset. Therefore, the sub-array remains precharged for the next cache access. Otherwise, the sub-array is not likely to be accessed and its bit-lines are isolated from the supply voltage source.

For data caches, different prediction techniques were proposed because the sub-array reference locality in data caches is lower than in instruction cache. In [48], a predictor indexed by memory reference instructions is used. Each entry contains the address of a memory reference instruction and its corresponding sub-array index. In [49], the same decay counter technique is used, but predecoding of the target sub-array is also employed to reduce the performance loss. The main idea of this predecoding technique is that for most of the memory instructions that use displacement addressing mode (*address = base address + displacement*), the accessed sub-array is determined by the base address.

However, the technique in [48] can incur a significant performance loss when the program counter repeatedly moves back and forth across the sub-array boundary in a loop. Furthermore, the performance impact of the technique in [49] is heavily reliant on the assigned threshold value, and, depending on application programs, the ideal threshold value differs. Determining it requires a very sophisticated algorithm to find an optimal threshold value for each class of application program.

2.3 Compiler Techniques

In [50, 51], various compiler-based cache leakage optimization strategies were introduced. Basically, the compiler-based leakage optimization techniques rely on the circuit techniques proposed in Section 2.1 or a technique that will be proposed in Chapter 3 of this thesis.

First, a compiler-based technique for instruction caches was proposed in [50]. It is based on determining the last usage of instructions. Once the last use of the instructions is detected, the corresponding cache lines are either turned off or switched to the state-preserving sleep state. The instruction cache lines are turned off at the loop granularity level. When loops are nested and the outer loop is exited, it is assumed that the inner loops will not be re-visited in the near future. The cache lines that hold the instructions belonging to the loop are either turned off or switched to the sleep mode. Based on this compile-time analysis, the compiler inserts the instructions controlling the power mode — active or stand-by — of the cache lines.

Second, a compiler technique for data caches was proposed in [51]. It is based on the observation that at a given time only a small fraction of the data cache lines need to be active (i.e., the lines that hold the currently used data) and the remaining cache lines can be placed into leakage-saving mode. Their approach activates a cache line just before the line is accessed as long as this can be known during the compile-time data-reuse analysis. After the access is completed, the cache line is turned off.

However, these approaches increase codes sizes due to the inserted cache control instructions and do not work very well when compiler optimization techniques for instruction caches, such as loop unrolling, are applied. In addition, this technique for the data caches is limited to array-based and pointer-intensive applications.

Chapter 3
Low-Leakage State-Preserving SRAM

The memory cell circuit proposed in this chapter employs a *dynamic voltage scaling* (DVS) technique to reduce leakage power. In *active mode*, a nominal supply voltage is provided to memory cells. However, when the cells are not intended to be accessed for a long period, they are placed in *sleep* or *drowsy mode*. In drowsy mode, a stand-by voltage in the range of 100~300*mV* is applied to the memory cells and the leakage power is significantly reduced due to the decrease in both leakage current and supply voltage [52].

Supply voltage reduction is especially effective for leakage power reduction due to the short-channel effects of *drain induced barrier lowering* (DIBL). This results in a super-linear dependence of leakage current on the supply voltage [53]. First, to understand the leakage power reduction achievable in drowsy mode, we present a sub-threshold leakage current analysis of memory cells. Second, we propose a leakage power reduction technique using dynamic voltage scaling. Finally, we discuss related issues and limits such as the lower bound of the stand-by supply voltage, wake-up latency and energy, cross-talk noise susceptibility, and soft-error sensitivity of the proposed circuit technique.

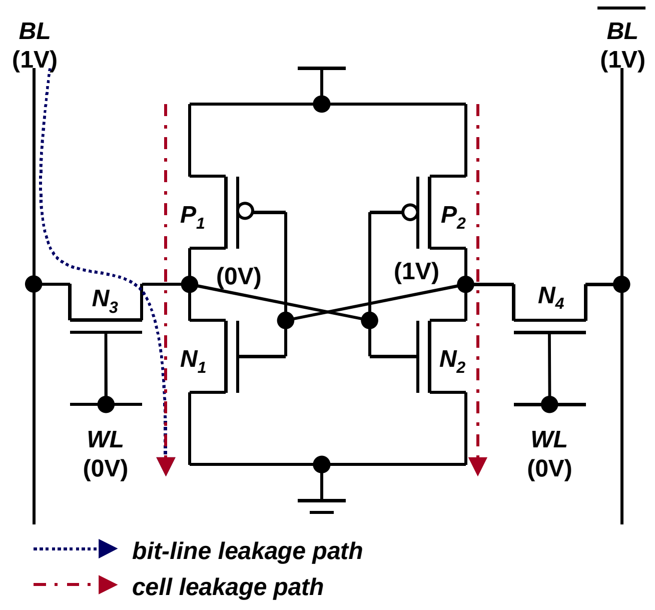
3.1 Leakage Analysis

To understand the leakage power reduction achievable in drowsy mode, a leakage modeling analysis is carried out in this section. Figure 3.1 shows a standard 6-transistor memory cell circuit. This contains the two leakage paths: 1) cell leakage; and 2) bit-line leakage. In the next generation 70*nm* technology, the leakage current in the off-state is dominated by the weak inversion current, which can be modeled as [52, 54]:

$$I_D = I_{s0} e^{(V_{GS} - V_{TH}) / \left(\frac{nkT}{q}\right)} \left(1 - e^{-V_{DS} / \left(\frac{nkT}{q}\right)}\right)(1 + \lambda V_{DS}) \qquad \textit{(Eq. 3-1)}$$

where λ is a parameter modeling the pseudo-saturation region in weak inversion. In Figure 3.1, we assume that P_1 and N_2 are in weak inversion while P_2 and N_1 are in strong inversion. When V_{DD} is high enough to keep P_2 and N_1 in strong inversion, the memory cell is strongly biased and the voltages V_{DD} and V_{SS} appear on the two cross-coupled

Figure 3.1: Leakage inside a biased static memory cell.



inverter nodes. Ignoring the current through N_3 and N_4, the overall leakage current is the sum of leakage currents from the off transistors. Here, the bit-line leakage through the access transistor N_3 and N_4 is not included in our modeling, because its contribution to the overall memory cell leakage is less than the cell leakage and it is dependent on the bit-line precharge technique, which varies widely among static random access memory designs. In addition, P_2 and N_1 are in the strong inversion region and only present a small serial resistance. Hence, they also can be ignored in the leakage modeling. Assuming $V_{GSP2} = V_{GSN1} = V_{SS}$ and $V_{GSP1} = V_{GSN2} = V_{DD}$, the overall leakage of the 6-transistor memory cell is modeled as following:

$$I_L = ((I_{SN} + I_{SP}) + (I_{SN}\lambda_N + I_{SP}\lambda_P)V_{DD})\left(1 - e^{-V_{DD}/\left(\frac{nkT}{q}\right)}\right) \qquad \text{(Eq. 3-2)}$$

where I_{SN} and I_{SP} are NMOS and PMOS off-transistor current factors that are independent of V_{DS} in Eq. 3-1.

3.2 Leakage Power Reduction using Dynamic Voltage Scaling

From Eq. 3-2, it can be seen that the leakage current decreases super-linearly with V_{DD}. Hence, significant leakage power can be reduced in drowsy mode, but a minimum voltage must be applied to memory cells to maintain correct memory states. Figure 3.2, shows the sub-threshold leakage power reduction of the memory cell implemented with a $70nm$ technology as supply voltage is scaled down. The memory cell dissipates $0.065\mu W$ at the $1V$ nominal supply voltage. However, we can reduce the leakage power of the memory cell by 92% at the $200mV$ stand-by supply voltage.

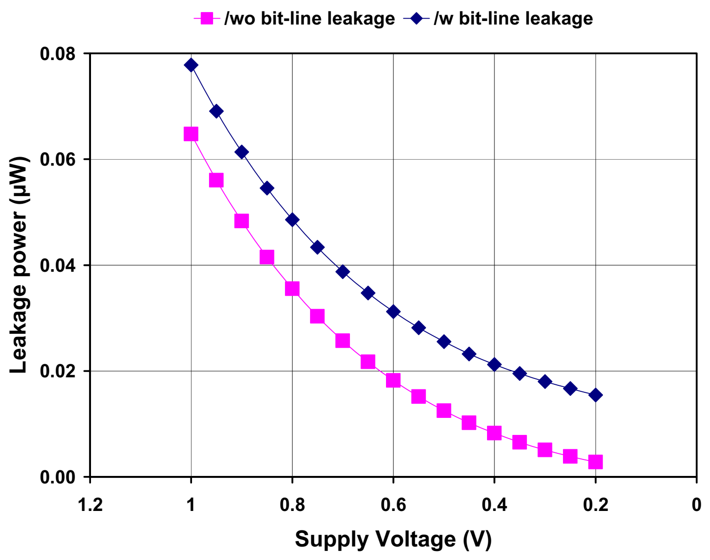


Figure 3.2: Sub-threshold leakage power reduction of the memory cell.

We initially ignored the bit-line leakage through the access transistors. However, total leakage power amounts to $0.072\mu W$ if the leakage by bit-lines is included. Compared with the cell leakage power, the bit-line leakage power through the access transistors is responsible for approximately 20% of the total leakage power of a memory cell in active mode. Furthermore, the dynamic voltage scaling technique barely reduces the bit-line leakage component as shown in Figure 3.2, because it is only effective for the cell leakage reduction.

To reduce the leakage power through the access transistors, either high-V_{TH} transistors should be used or bit-line precharge circuits should be turned off. However, the former technique increases the memory access time, because high-V_{TH} transistors are slower than low-V_{TH} ones. The latter requires an additional cycle to initiate a new precharge cycle. The precharge clock signals are asserted every cycle because bit-line precharging and address decoding should be performed concurrently in conventional precharge techniques.

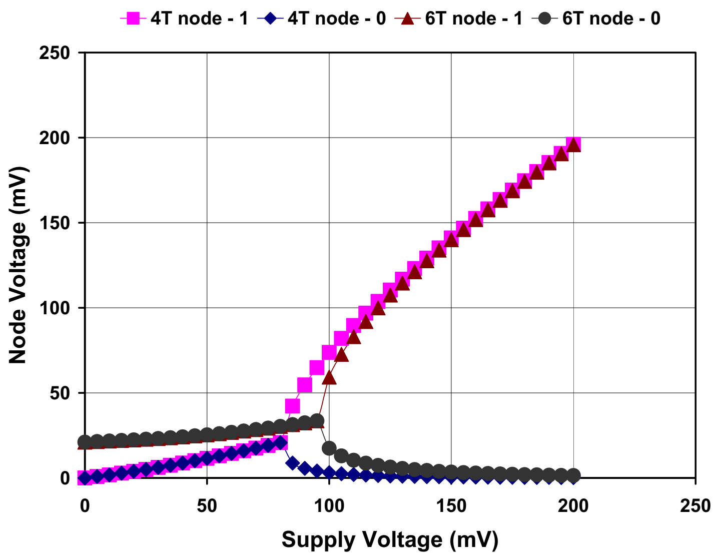


Figure 3.3: Minimum data retention voltage of the memory cell.

We sweep the supply voltage from 1 to 0*V* using HSPICE to derive a minimum state-preserving voltage as shown in Figure 3.3 where “4T node - 1” and “6T node - 1” represent nodes holding logic state “1” in a cross-coupled inverter pair consisting of 4 transistors (P_1, N_1, P_2, and N_2 in Figure 3.1) and a complete memory cell of 6 transistors including access transistors (N_3, and N_4), respectively. The corresponding case for the nodes holding the logic state “0” is also shown. As the supply voltage is scaled down, the voltage of both nodes approaches the same level and eventually, they become indistinguishable below 100*mV* range. This implies that the memory cell state has been destroyed.

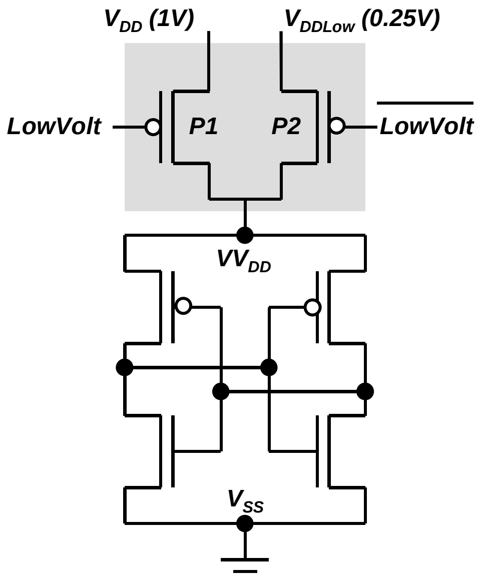
To keep a meaningful state, one cross-coupled node should maintain logic “1” while the other should hold logic “0” in the memory cell. However, there is no way to recover the original logic state once the voltage levels of both nodes become indistinguishable. Therefore, the stand-by voltage must be higher than the minimum state-preserving voltage. Furthermore, it should include a margin for process variations such as V_{TH} and transistor channel length.

3.3 Low-Leakage and State-Preserving SRAM Architecture

Figure 3.4 illustrates our proposed low-leakage, state-preserving drowsy memory cell with a supply voltage control mechanism. Based on operating mode — active or drowsy, the two PMOS transistors, P_1 and P_2, control the supply voltage of the memory cells connected to the same power supply rail. When the memory cells are in active mode, P_1 supplies the nominal supply voltage (V_{DD}), and P_2 provides the stand-by voltage (V_{DDLow}). P_1 and P_2 are controlled by complementary supply voltage control signals (*LowVolt* and $\overline{LowVolt}$). In addition, there will be negligible supply voltage degradation at the VV_{DD} node, because PMOS transistors transmit the full supplied voltage unlike NMOS transistors, which cause a V_{TH} *drop*.

In drowsy mode, however, the memory cell accesses are not allowed, because the voltage level of the bit-lines is higher than that of the cross-coupled inverter core node, which may result in loss of the memory cell state. Moreover, the reduced supply voltage

Figure 3.4: A memory cell with dynamic voltage scaling mechanism.



may cause the sense-amplifiers to operate incorrectly, because the memory cells may not have enough current driving capability to sink the charge stored in the bit-lines.

3.3.1 Wake-up Latency and Energy

When the memory cells are in drowsy mode, it takes a finite amount of time to restore the voltage level of the VV_{DD} node from stand-by to nominal supply voltage level, which we refer as “wake-up” latency. Furthermore, it consumes dynamic energy during the stand-by to active mode change. The drowsy cache line wake-up latency is critical for L1 caches because it increases the access latency when the processor accesses the lines. A fast wake-up latency is desirable to minimize the performance loss. Furthermore, the leakage saving during stand-by mode must be able to compensate for the wake-up energy overhead. In fact, the leakage savings must be larger than the wake-up energy overhead.

In a low-leakage memory technique such as adaptive-body biasing, wake-up is very slow and energy hungry, because the wake-up circuitry drives the body or substrate, which has a significant amount of the capacitance. Therefore, this technique is not suitable for L1 caches, which require a fast wake-up with little energy dissipation. The gated-ground technique for L1 caches destroys the memory cell states when it is put into leakage saving mode. Turning off a wrong cache line that will be accessed again incurs extra L2 cache accesses resulting in more dynamic energy dissipation and more penalty cycles.

To estimate the wake-up latency, we connected 128 memory cells to the power supply rail and the voltage control circuitry shown in Figure 3.4. In deep sub-micron technologies, the delay of the interconnect wires is typically more significant than that of the gates. To account for this, we modeled the interconnect capacitance and resistance based on the estimated power supply rail wire length and width using the 70*nm* technology

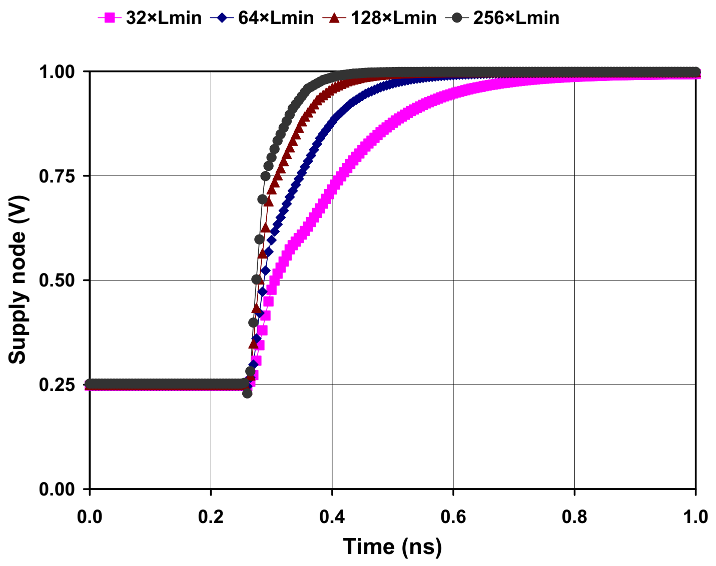


Figure 3.5: Wake-up latency of VV_{DD} node.

parameters [4]. To estimate a memory cell dimension with 70*nm* technology, we applied a linear scaling to an Artisan™ 0.18μm technology memory cell. The detailed power supply rail wire dimension and interconnect parameters for the HSPICE simulations are summarized in Table A.1 and Table A.2, respectively.

Figure 3.5 shows the wake-up latencies of the VV_{DD} node as the width of the P_1 transistor in the voltage control circuitry shown in Figure 3.4 is increased. We assumed 1*V* and 0.25*V* for the nominal and stand-by supply voltages, respectively. To estimate the number of cycles required for restoring the supply voltage level of VV_{DD} node, we need to estimate the clock frequency of a typical processors. According to [55, 56], the cycle time of the high-end microprocessors has been around 16×FO4 (fan out of four) delay. This corresponds to 527*ps* in 70*nm* technology, and it will approach 12×FO4 in future technology [57].

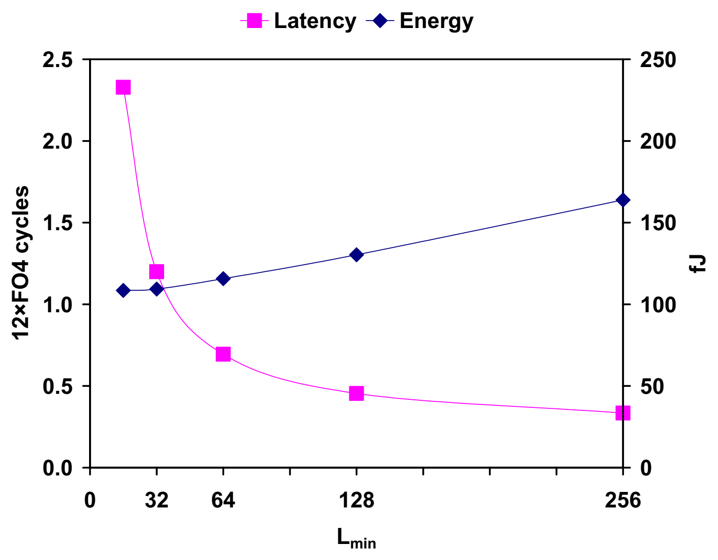


Figure 3.6: Wake-up latency and energy of the voltage control transistors.

Table 3.1 and Figure 3.6 show the wake-up latency and energy estimated for each size of the voltage control transistor. We defined the VV_{DD} rise time from 0.25V to 0.99V as a wake-up latency. The latency number in the parenthesis is normalized to the 12×FO4 delay. The energy number in Table 3.1 includes the dissipation by the circuitry driving the voltage control transistor. When we use 32× and 64×L_{min} size voltage control transistors, we can restore the full supply voltage level of the VV_{DD} node in 1 or 2 cycles, respectively. Considering the aggressive assumption on the clock cycle time, the normalized numbers of cycles in Table 3.1 are conservative. However, the voltage control transistor supplying stand-by voltage (P_2 in Figure 3.4) does not need to be such a wide PMOS transistor (e.g.,

Table 3.1: Wake-up latency and energy of the voltage control transistors.

	16×L_{min}	**32×L_{min}**	**64×L_{min}**	**128×L_{min}**	**256×L_{min}**
Latency (*ps*)	919 (2.33)	473 (1.20)	274 (0.69)	179 (0.45)	132 (0.33)
Energy (*fJ*)	108	109	116	130	164

L_{min} is equivalent to 2λ and the number in the parenthesis is normalized to 12×FO4 delay.

$64 \times L_{min}$ size), because the latency from active to drowsy mode is not critical for processor performance. Thus, we can use a minimum size transistor that provides enough current to sustain the stand-by voltage level of the cache lines.

To estimate the extra area used by the voltage control circuitry, we extracted an actual cell layout from a memory compiler with the TSMC $0.18\mu m$ design rules. This technology was the smallest feature size available to the academic community at the time of the experiment. The dimensions of the memory cell are $3.66\mu m$ (40λ) $\times$ $1.84\mu m$ (20λ), and those for the voltage control circuitry are $3.66\mu m$ (40λ) $\times 1.98\mu m$ (22λ) and $3.66\mu m$ (40λ) $\times 3.42\mu m$ (38λ) for $32\times$ and $64 \times L_{min}$ size voltage control transistors, respectively. We estimate the extra area by the voltage control circuitry per 128-bit cache line is equivalent to 1.1 (0.87%) and 1.9 (1.48%) extra memory cells for the $32\times$ and $64 \times L_{min}$ voltage control transistors, respectively. This relatively low area overhead can be achieved because of the negligible interconnect overhead in the voltage control circuitry compared to the 6-transistor memory cell. In fact, a significant amount of the memory cell area is consumed by local interconnects and spacing between different transistors even though the actual size of each transistor is small.

3.3.2 Cross-Talk Noise and Soft-Error Susceptibility

In ultra deep sub-micron technology, the cross-coupling capacitance between interconnects or nodes are becoming significant. On the other hand, we can suppress more leakage power as we decrease the supply voltage. However, reducing the supply voltage also decreases the charged stored in the nodes. This will make the memory cells in stand-by mode more susceptible to *cross-talk noise* [58] and *soft-errors* caused by cosmic radioactive particles [59].

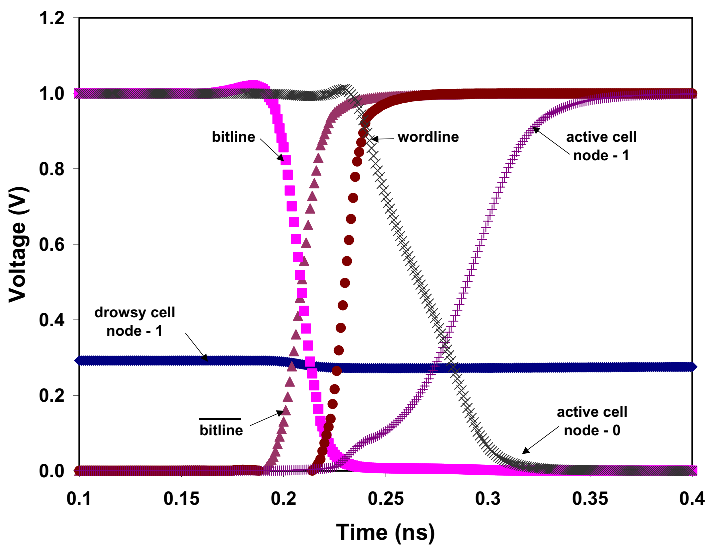


Figure 3.7: Cross-talk noise stability of the drowsy cell.

We examined the cross-talk noise susceptibility of the drowsy cells in a row by applying a write operation to the adjacent row that is in active mode. This makes all the bit-lines, connecting both drowsy and active cells, swing rail-to-rail. According to the HSPICE simulation shown in Figure 3.7, there is only a slight voltage fluctuation at the drowsy node - 1 during the write operation to the active row. The voltage level of the drowsy cell recovers its stand-by voltage level quickly because the cross-coupled inverters in the drowsy cell keep driving their internal nodes.

We next consider soft error susceptibility. To avoid soft errors, the collected charge Q at that particular node should be more than $Q_{critical}$. If the charge generated by a particle strike at the node is more than $Q_{critical}$, the pulse generated is latched at the node, and results in a bit flip. This concept of critical charge is generally used to estimate a *soft-error rate* (SER). In [60], a technique to estimate the SER in CMOS memory circuits was developed. In this model an exponential dependence of SER on critical charge was shown as following:

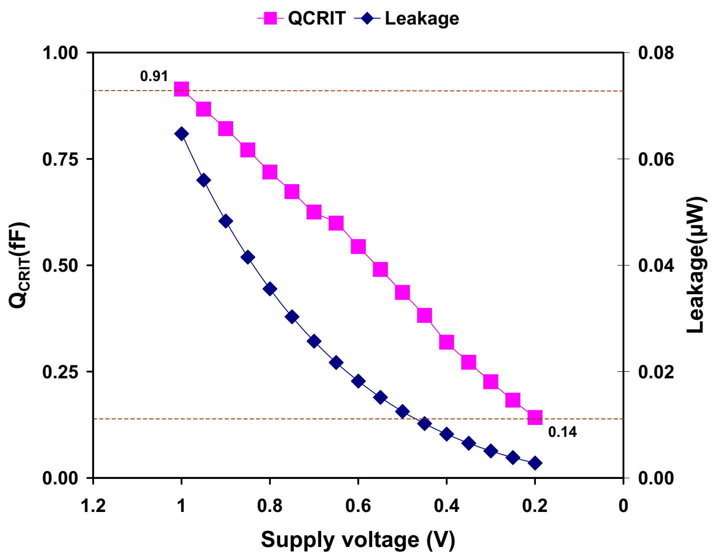


Figure 3.8: $Q_{critical}$ of a drowsy cell.

$$SER \propto N_{flux} \times CS \times \exp\left(\frac{-Q_{critical}}{Q_s}\right) \qquad \text{(Eq. 3-3)}$$

where N_{flux} is the intensity of the neutron flux, CS is the area of the cross section of the node, and Q_s is the charge collection efficiency depending on the doping concentration.

The value of $Q_{critical}$ is proportional to the node capacitance and the supply voltage. Hence, $Q_{critical}$ at a node will decrease as the supply voltage or node capacitance is reduced. Assuming that the node capacitance is fixed for the same memory cell, SER is only proportional to $Q_{critical}$. Figure 3.8 shows the $Q_{critical}$ and leakage power as the supply voltage decreases. To measure $Q_{critical}$, we applied a current pulse to a memory cell core node for a fixed period of time increasing the magnitude of the current until the bit flips. At the flipping point, we measured the magnitude of the current and integrated the current for that time period. As seen in Figure 3.8, $Q_{critical}$ decreases linearly with the supply voltage.

This shows that memory cells in stand-by mode are more susceptible to soft-errors. However, we can employ an *error detection code* mechanism to deal with these errors. In addition, we can fix errors by fetching correct memory states from the L2 cache or the external memory as soon as a soft-error is detected in the L1 caches. By using the L2 cache for correction we can avoid the need for more complex error correction codes in the L1 caches.

3.4 Chapter Summary

In this chapter, we proposed a new low-leakage state-preserving static random access memory circuit architecture. It is simple but effective at reducing the memory cell leakage power and has fast wake-up latency and little energy dissipation. It reduces the memory cell leakage power by 94% at 250*mV* stand-by voltage with 1 or 2 cycles wake-up latency depending on the transistor size of the voltage control circuitry.

The state-preserving capability, fast wake-up latency, and low energy dissipation during wake-up, allows the computer architect to employ more aggressive L1 cache leakage management policies. In the following sections, we will introduce the microarchitectural control policies that manage the L1 caches with the drowsy memory architecture proposed in this chapter.

Chapter 4
Drowsy Data Caches

Over a fixed time period, it has been observed that the activity in a data cache is only centered on a small sub-set of the cache lines. This behavior can be exploited to cut the leakage power of large data caches by putting the *cold* cache lines into a state preserving, low-power *drowsy* mode. In the state-preserving drowsy caches, the cost of being wrong — putting a line into drowsy mode that is accessed soon thereafter — is relatively small compared to the caches that use the state-destroying *gated*-V_{DD} technique.

Although the leakage saving of the state-preserving technique is slightly less than that of the state-destroying one, the only penalty one must contend with is an additional delay and energy cost for having to wake up a drowsy line. One of the simplest policies that one might consider is that all lines in the cache — regardless of access patterns — are put into drowsy mode periodically and each line is woken up only when it is accessed again. This policy requires only a *single global counter* generating the periodic drowsy signal, and no *per-line* statistics [44].

In this chapter, we investigate microarchitectural control mechanisms for putting L1 data cache lines into their drowsy state and compare them to one another. We argue that the simple policy of periodically putting the entire cache lines into drowsy mode does about as well as a much more complex control policy that tracks accesses to each cache line.

4.1 Characterizing Working Sets

Table 4.1 shows the working set characteristics of a sub-set of SPEC2K benchmarks for a 32KB 2-way set associative data cache with 1024 32-byte lines. The observations of cache activity are made over 2048 cycle periods, which we refer to as 2048 cycle update windows. (See Appendix B and Table B.1 for the detailed simulation methodology and parameters, respectively.) The table shows that on most of the workloads, the working set — the fraction of unique cache lines accessed during an update window (e.g., 2048 cycles) — is relatively small. On average, only 12% of the 1024 32-byte cache lines need to be in active mode, and the remaining lines can be in drowsy mode at any one time. This

Table 4.1: Data cache working set characteristics.

benchmark	working set	number of accesses	accesses / line	accesses / cycle
bzip2	7%	820.5	10.9	0.40
crafty	16%	1313.9	8.0	0.64
gcc	34%	2898.8	8.4	1.42
parser	8%	894.4	10.3	0.44
vortex	11%	1319.6	11.4	0.64
ammp	8%	680.3	8.6	0.33
equake	6%	1956.2	29.7	0.96
fma3d	5%	787.6	14.1	0.38
mgrid	16%	907.2	5.6	0.44
swim	12%	368.2	2.9	0.18
AVG	12%	1082.2	8.9	0.40

A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used.

has the potential to significantly reduce the static power consumption of the cache. The downside of the approach is that the wake-up cost has to be amortized over a relatively small number of accesses between 2.9 (*swim*) and 29.7 (*equake*), depending on the workloads. (See Table B.2 for the working set characteristics of the entire SPEC2K benchmark suite.)

The *execution factor* (EF) computing the expected *worst-case execution time increase* for the simple algorithm can be formulated by:

$$EF = \frac{accesses \times \left(\frac{wake\text{-}up\ latency \times memory\ impact}{accesses\ /\ line}\right) + window\ size}{window\ size} \qquad (Eq.\ 4\text{-}1)$$

All variables except *memory impact* and *wake-up latency* are directly from Table 4.1. The term memory impact can be used to describe how much impact a single memory access has on overall performance. The simplifying assumption is that any increase in cache access latency translates directly into increased execution time, in which case the memory impact is set to 1. Using this formula and assuming a 1 cycle wake-up latency, we get a maximum of 17% performance degradation for *gcc* and under 4% for *equake*.

One can further refine the model by coming up with a more accurate value for memory impact. Its value is a function of both the microarchitecture and the workload:

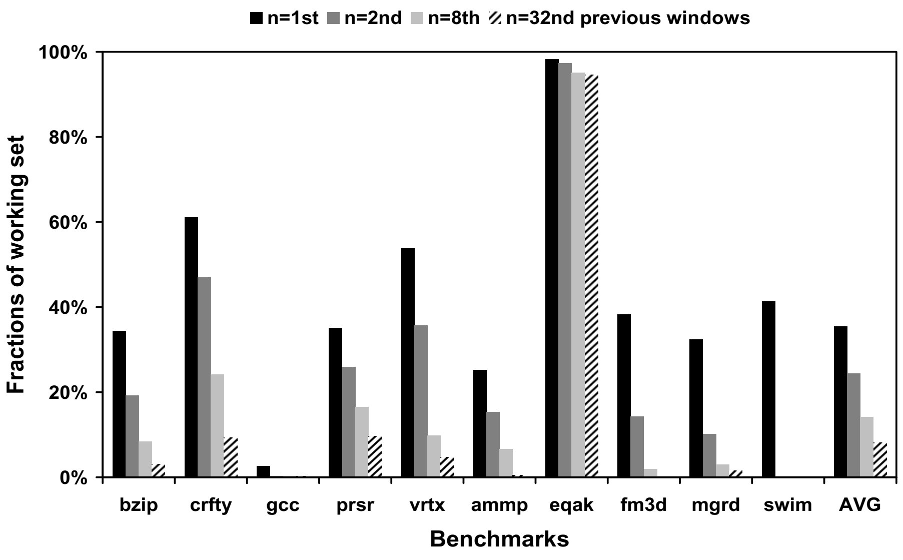
- The workload determines the ratio of the number of memory accesses to instructions.
- The microarchitecture determines what fraction of wake-up transitions can be hidden, i.e., not translated into global performance degradation.

- The microarchitecture also has a significant bearing on *instruction per cycle* (IPC) which in turn determines the number of memory accesses per cycle.

Assuming that half of the wake-up latencies can be hidden by the microarchitecture, and based on a ratio of 1.42 of memory accesses per cycle, the prediction for worst-case performance impact for *gcc* reduces to 12%. Similarly, using the figure of 0.96 memory accesses per cycle and the same fraction of hidden wake-up transitions, we get a performance impact of about 1.5% for *equake*. The actual impact of the baseline technique is likely to be significantly lower than the results from the analytical model, but nonetheless, these results suggest that there is no need to look for prediction techniques to control the drowsy cache; as long as the drowsy cache can transition between drowsy and awake modes relatively quickly, simple algorithms should suffice.

Figure 4.1 shows the data cache working set reuse characteristics — the fraction of accesses that are the same as in the *n-th* previous window for the sub-set of SPEC2K benchmarks. (See Table B.3 for the statistics of all the SPEC2K benchmarks.) The results in the figure specify what fraction of references in a current window are to lines that had been accessed 1, 2, 8, or 32 windows before. This information can be used to gauge the applicability of control policies that predict the working set of applications based on past accesses. As it can be seen, on many workloads (e.g., *bzip* and *gcc*), a significant fraction of lines are not accessed again in a successive window. This implies that past accesses are not always a good indication of future use.

Figure 4.1: Data cache working set reuse characteristics.



A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used. The fractions of accesses that are the same as in the n-th previous window in the current 2K cycle window size.

Aside from *equake* where past accesses do correlate well with future accesses, most workloads only re-access 40% of the lines between windows and the fractions from the previous windows decrease dramatically as cycles go on. The implications of this observation are twofold: If an algorithm keeps track of which cache lines are accessed in a window, and only puts the ones into drowsy mode that have not been accessed in a certain number of past windows, then the number of awake to drowsy transitions per window can be reduced by about 50%. This, in turn, decreases the number of later wake-ups, which reduces the impact on execution time. However, the impact on energy savings is negative because a larger fraction of lines are kept in full power mode, and in fact many of those lines will not be accessed for the next several windows, if at all.

Table 4.2: Latencies of accessing a cache line in the drowsy cache.

	Active	Drowsy
Hit	• 1 cycle	• 1 cycle — to wake-up line • 1 cycle — to read / write line
Miss	• 1 cycle — to find line to replace • *n* cycle — to access upper memory hierarchy	• 1 cycle — to find line to replace ~~• 1 cycle — to wake-up line~~ • *n* cycle — to access upper memory hierarchy

The 1-cycle wake-up latency shown with a strike-through is overlapped with the upper memory access latency.

Table 4.2 shows the latencies associated with the different modes of operation. No extra latencies are involved when an active line is accessed. Hits and misses are determined the same way as in normal caches for the active cache lines. However, a hit for the drowsy cache line costs one extra cycle to wake up the line although this wake-up penalty may be overlapped with the upper memory such as L2 or main memory access latency.

4.2 Policy Evaluation

In this section, we evaluate the different policies with respect to their impact on run-time and the fraction of cache lines that are in drowsy mode during the execution of SPEC2K benchmarks. The following parameters can be varied:

- The *update window size* specifies in cycles how frequently decisions are made about which cache lines are put into drowsy mode.
- The *wake-up latency* is the number of cycles for waking up drowsy cache lines. We consider 1, 2, or 4 cycle transition times since our circuit simulations indicate that 1 or 2 cycles are reasonable assumptions — with 4 cycles being a conservative extreme.

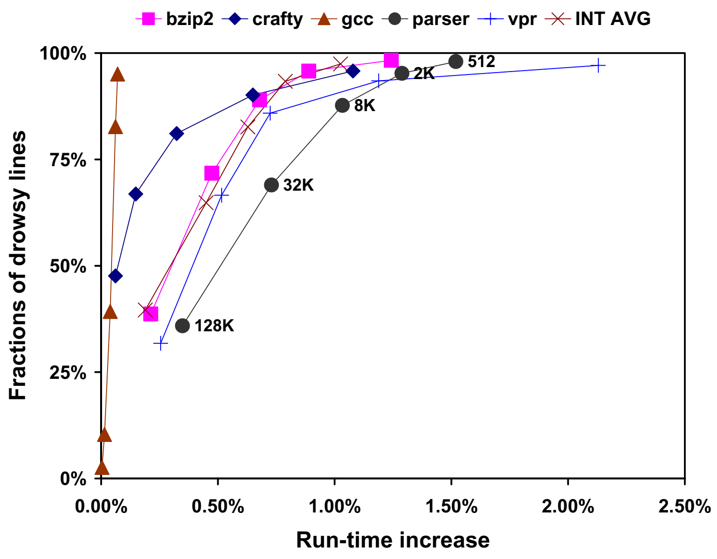
- The policy that uses no per-line access history is referred to as the *simple* policy. In this case, all lines in the cache are put into drowsy mode periodically; the period is the window size. The *noaccess* policy means that only lines that have not been accessed in a window are put into drowsy mode.

The detailed simulation methodology and processor parameters are discussed in Appendix B and Table B.1.

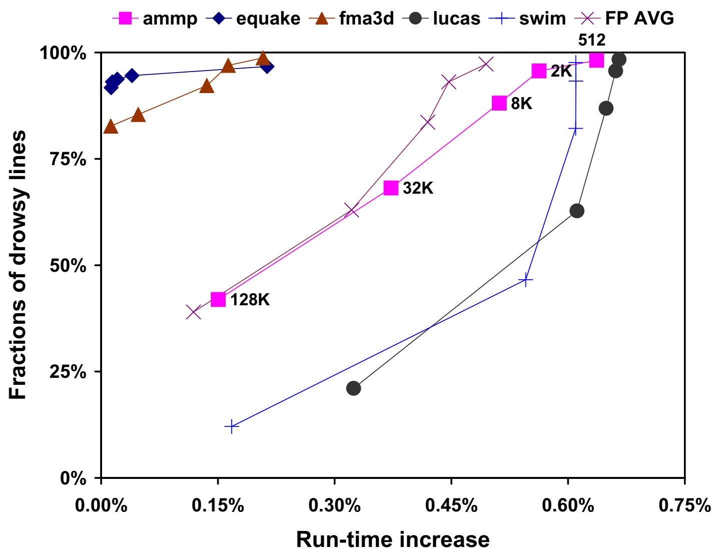
Figure 4.2 shows how the update window size impacts the run-time and the fraction of drowsy lines using a simple policy with an 1-cycle wake-up latency. For clarity, we are showing only a sub-set of the benchmarks with an average from the entire benchmarks. As we decrease the update window size, we have a larger fractions of drowsy lines, which implies that we are able to reduce more leakage power dissipation. However, this causes run-time increases for the workloads. Also, at the same update window size, the floating-point workloads show less run-time increase compared to the integer ones, because the floating-point applications have more temporal locality — accesses remain in a specific region of the data cache longer according to our experimental results. On the give processor configuration, the sweet-spot — where the energy-delay product is maximized — is around 2K cycles for the simple policy with a 1-cycle wake-up latency.

From the entire SPEC2K benchmarks, the average fractions of drowsy lines are 97%, 93%, 83%, 64%, and 39%, while the average run-time increases are 0.76%, 0.62%, 0.52%, 0.39%, and 0.15% for 512, 2K, 8K, 32K, and 128K update window sizes. The reason for the relatively small impact of the drowsy wake-up penalty on the processor's performance is due to our use of a non-blocking memory system, which can handle a number

Figure 4.2: Impact of window sizes on the run-time and fraction of drowsy lines.



(a) SPEC2K integer benchmarks



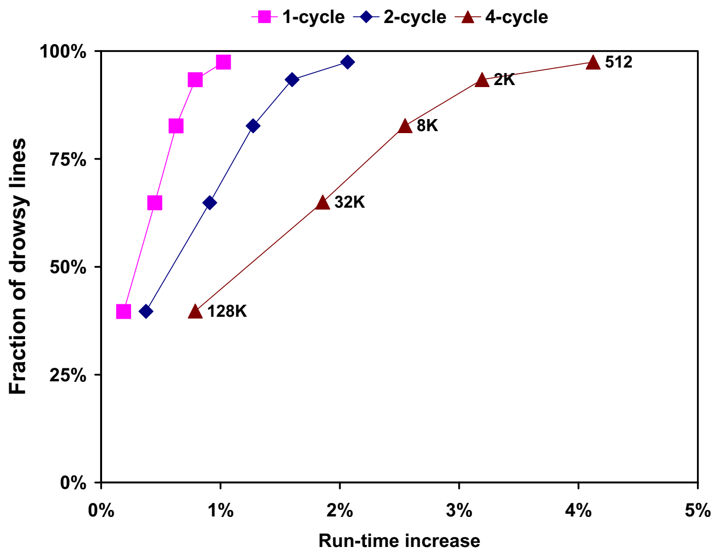
(b) SPEC2K floating-point benchmarks

We use a simple policy with 128K, 32K, 8K, 2K, and 512 update window sizes and 1-cycle drowsy-line wake-up latency.

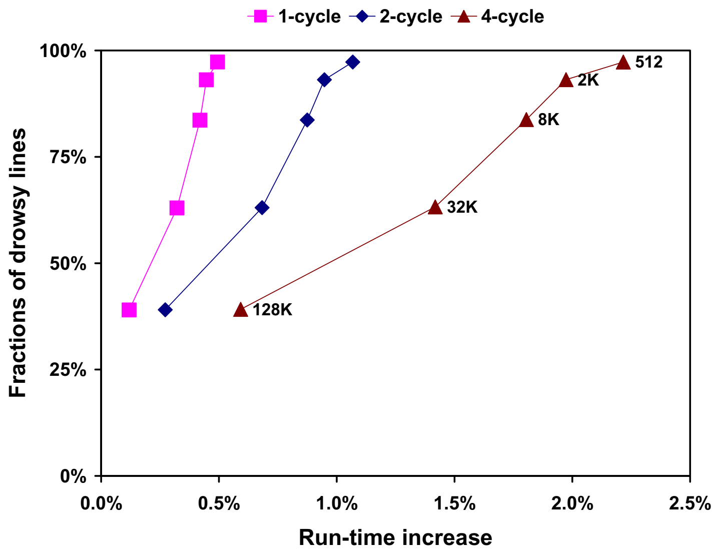
of outstanding loads and stores while continuing execution of independent instructions. Moreover, the drowsy wake-up penalty is usually only incurred with load instructions, because stores are put into a write buffer, which, if not full, allows execution to continue without having to wait for the completion of the store instruction. In terms of the average leakage power reduction, roughly 85% of leakage power can be reduced with a 0.62% run-time increase when the average fraction of drowsy lines are 93%, and a drowsy cache line consumes 10% of the leakage power of an awake line.

The impact of increased wake-up latencies is shown in Figure 4.3. The graphs in the figures show the run-time increases of the processor using a simple policy with 1-, 2-, and 4-cycle wake-up latencies and 512, 2K, 8K, 32K, and 128K update window sizes. According to the experimental results, the fraction of drowsy lines remain relatively constant, but the run-time increases linearly as the wake-up latency is increased, because the extra number of wake-up cycles increases idle cycles in the processor pipeline. For the 512, 2K, 8K, 32K, and 128K update window sizes, the average run-time increases of the simple policy with the 2-cycle (4-cycle) wake-up latency are 1.6% (3.2%), 1.3% (2.6%), 1.1% (2.2%), 0.8% (1.6%), and 0.3% (0.7%), respectively. The impact on the run-time is doubled as the wake-up latency is doubled and the run-time increase trend is consistent with the results of 1-, 2-, and 4-cycle wake-up latencies. To minimize the run-time impact of the wake-up penalty, it is necessary to use a voltage controller that wakes up the drowsy cache line as fast as possible, but this increases the area overhead as well as dynamic power dissipation of the voltage controller — see Section 3.3.1 for the wake-up latency and energy for the controller. However, the cost of the extra area and energy needed for the voltage controller is relatively small. Therefore, we will use 1 cycle as a wake-up latency.

Figure 4.3: Impacts of increased drowsy access latencies.



(a) SPEC2K integer benchmarks



(b) SPEC2K floating-point benchmarks

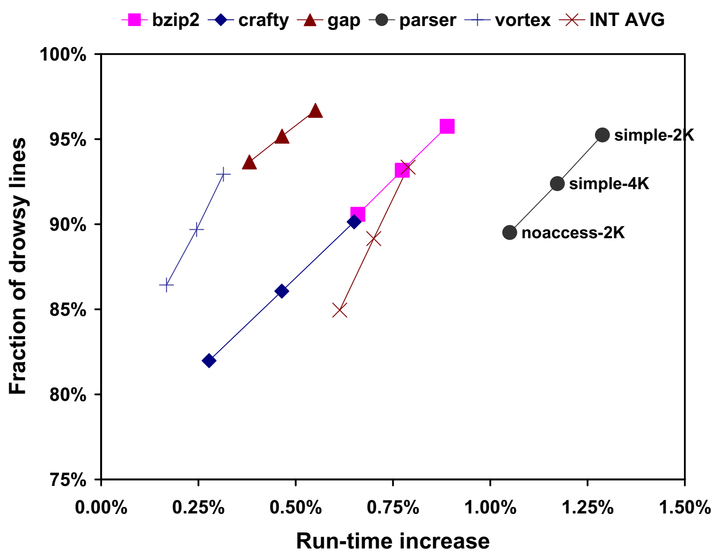
We use a simple policy with 1-, 2-, and 4-cycle drowsy-line wake-up latencies and 512, 2K, 8K, 32K, and 128K update window sizes.

Figure 4.4 contrasts the noaccess and the simple policies. The main question that we are trying to answer is whether there is a point to keeping any per-line statistics to guide drowsy decisions or if the indiscriminate approach is good enough. We show three different configurations for each workload on the graph: the noaccess policy with a 2K-cycle window and two configurations of the simple policy (4K- and 2K-cycle windows). In all cases, the policy configurations follow each other from bottom to top in the afore-mentioned order. This means that in all cases, the noaccess policy has the smallest fraction of drowsy lines, which is to be expected, since it is conservative about which lines are put into drowsy mode.

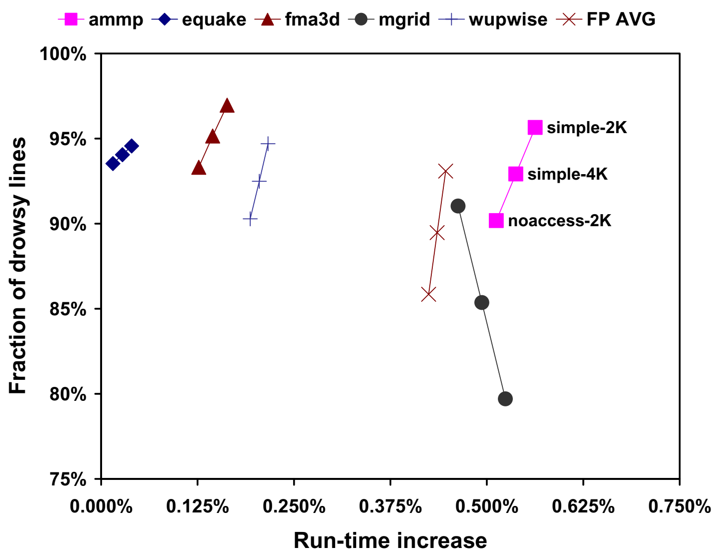
The workloads on the graph can be partitioned into two groups: ones on lines whose slopes are close to the vertical, and ones on lines that are more horizontal and thus have a smaller positive slope. All the workloads that are close to the vertical are floating point workloads and their orientation implies that there is very little or no performance benefit to using the noaccess policy or larger window sizes. In fact, *mgrid* in the graph has a slight negative slope, implying that not only would the simpler policy win on power savings, it would also win on performance. However, in all cases the performance difference is negligible and the potential leakage power improvement is under 5% in most floating point applications. The reason for this behavior is the very bad reuse characteristics of data accesses in these workloads. Thus keeping lines awake (i.e., noaccess policy, or larger window sizes) is unnecessary and even counterproductive.

This anomalous behavior is not replicated on the integer workloads, where in all cases the noaccess policy wins on performance, but saves the least amount of power. Does this statement imply that if performance degradation is an issue then one should go with

Figure 4.4: Comparisons of noaccess and simple policies.



(a) SPEC2K integer benchmarks



(b) SPEC2K floating-point benchmarks

The bottom markers on each line corresponds to the noaccess policy with 2K-cycle window, the markers above it represent the simple policy with 4K- and 2K-cycle windows, respectively.

the more sophisticated noaccess policy? It does not. The slope between the upper two points on each line is almost always the same as the slope between the bottom two points, which implies that the rates of change between the data points of a workload are the same; the data point for the noaccess policy should be able to be matched by a different configuration of the simple policy. We ran experiments to verify this hypothesis and found that a window size of 8K of the simple policy comes very close to the coordinates for the noaccess policy with a window size of 2K. We find that the simple policy with a window size of 4K cycles reaches a reasonable compromise between simplicity of implementation, power savings, and performance. The impact of this policy on leakage energy is evaluated in Section 4.4.

4.3 Circuits for Microarchitectural Drowsy Cache Controls

Figure 4.5 shows the circuits to support microarchitectural control of drowsy caches with the circuit technique proposed in Section 3.3. There are a few additions: a *voltage controller*, a *drowsy set/reset flip-flop*, and a *word-line gating circuitry* for each standard cache line. To support the drowsy mode, a voltage controller is connected to the power supply rail (VV_{DD}) of each cache line. This switches the supply voltage of the cache line between the nominal (active) and stand-by (drowsy) supply voltages depending on the state of the drowsy set/reset flip-flop — see Figure 4.5 for the signal interconnections between the components.

The drowsy set/reset register is a special flip-flop supporting both *asynchronous* and *synchronous set/reset*. (See Figure 4.6 for the circuit schematic.) It is reset by a *global periodic drowsy signal* and set by a *signal* from the *final decoder* or *pre-buffered word-*

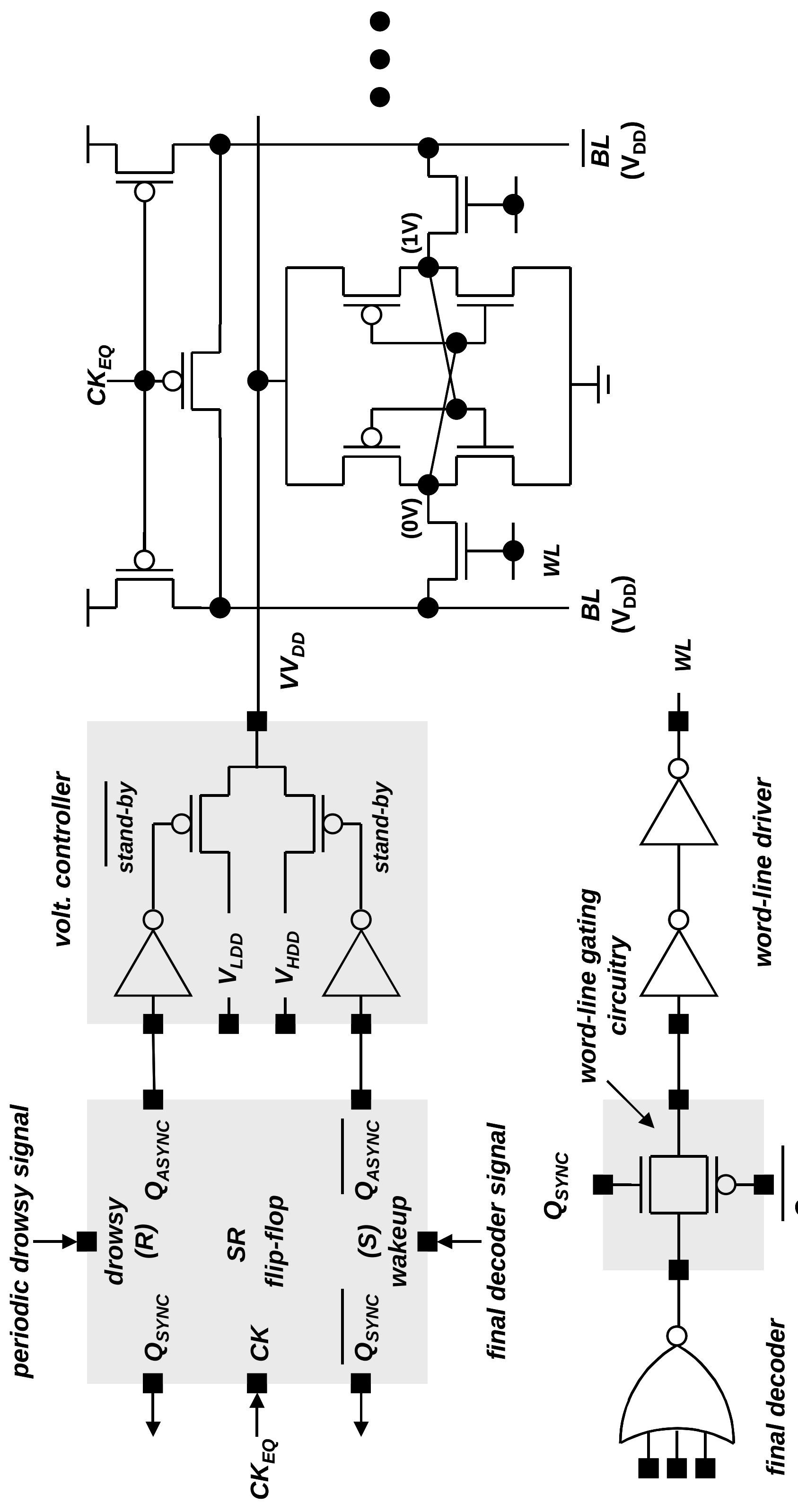


Figure 4.5: Circuits to support microarchitectural control of a drowsy cache line.

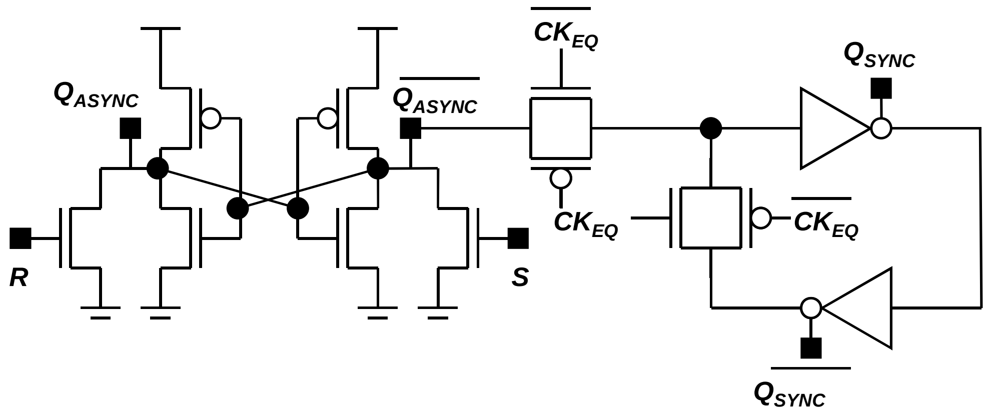


Figure 4.6: Drowsy set/reset flip-flop circuitry.

line signal — see Figure 4.5-(a) for the interconnections. Q_{ASYNC} and $\overline{Q}_{ASYNC}$ reflect the immediate change of the register state by the set and reset signals.) These asynchronous outputs are required to switch mode of the voltage controller immediately as soon as either the global periodic drowsy or pre-buffered word-line signal is asserted. However, Q_{SYNC} and $\overline{Q}_{SYNC}$ changes effectively only at the negative edge of the pre-charge clock signal (CK_{EQ}), because both activating the word-line gating circuitry and reading the drowsy register state must be synchronized to the pre-charge clock.

The word-line gating circuitry between the final decoder and the word-line driver is used to prevent accesses of a cache line in drowsy mode, because the supply voltage of the cache line in drowsy mode is lower than the pre-charged bit-line voltage; unchecked accesses to the drowsy mode cache line may destroy the memory states. The assertion of a pre-buffered word-line signal sets the drowsy register. However, the signal should not be propagated to the word-line driver in the current cycle, because the cache line has not been woken up yet. In the pulsed word-line technique, the pre-buffered word-line signal is deactivated before the negative edge transition of the pre-charge clock signal, and the $\mathrm{Q_{SYNC}}$ and $\overline{\mathrm{Q}}_{\mathrm{SYNC}}$ reflect the flip-flop state after the negative edge transition of the clock.

Hence, the word-line signal is not propagated in the current cycle, but in the consecutive cycle when the cache line is awakened.

Whenever a cache line is accessed, the cache controller monitors the condition of the voltage of the cache line by reading the drowsy flip-flop (Q_{SYNC} and $\overline{Q}_{SYNC}$) that is always maintained by the nominal supply voltage. If the accessed line is in active mode, we can read out the cache line without any loss of performance. No performance penalty is incurred, because the power mode of the line can be checked by reading the drowsy register concurrently with the read-out and comparison of the tag. However, if the memory array has been in drowsy mode, the cache line will be woken up automatically in the next cycle, and the data can be read out during consecutive cycles.

4.4 Run-time Impact and Leakage Power Reduction

To examine the effectiveness of the proposed technique, we compare our drowsy cache techniques with a state-of-art *cache decay* technique [44]. In the cache decay technique, it is critical to consider the energy penalty from extra L2 cache accesses, because decayed or dead L1 cache lines result in additional accesses to the L2 cache. We assume that a 512KB 4-way set associative L2 cache is designed with the sub-banking technique of [5] to improve both the access time and energy dissipation efficiency of the cache. As a L2 cache access latency, we use 12 cycles per access. We also estimated the energy dissipation of the L2 cache (653*pJ* per access) using our modified version of CACTI 3.2 [61] assuming a 70*nm* technology and the number of sub-banks that result in the least energy dissipation per access.

To benchmark the two techniques, we compare the normalized leakage energy. This is the ratio of total leakage energy dissipation by the L1 cache (plus dynamic energy dissipation from the extra L2 cache accesses for the cache decay technique), divided by the total leakage energy consumed in the same size regular cache. To calculate the leakage energy from the leakage power, we need to estimate the cycle time of the processor. We again use the 12×FO4 delay of Section 3.3.1. The HSPICE simulation with the projected 70*nm* technology shows that 12×FO4 is around 395*ps*.

Table 4.3 shows the comparison of the processor run-time impact and the leakage power reduction between the drowsy data cache and cache decay techniques. We use the simple policy with a 4K-cycle drowsy window size and 1-cycle wake-up latency for the drowsy data cache and the 8K-cycle decay window size for the cache decay technique; we also swept the decay window size to find the optimal window size and we found out that the optimal window size is 8K, which is the same as that in [44]. The normalized leakage figures in the parenthesis are re-calculated using the leakage power of the 6-transistor static memory cell implemented with 300*mV* high-V_{TH} access transistors. For all the results in the table, we conservatively assume that there are only 20 tag bits for the 32KB 2-way set associative cache (corresponding to 32 bit addressing) per line, which translates into 6.9% of the bits in a cache line[1]. The experimental results show that our implementation of a drowsy data cache can reduce the total leakage energy consumed in the data cache by 65% without significant run-time impact (0.57% in average). If the memory cell is implemented with 300*mV* high-V_{TH} access transistors, the leakage energy could potentially be reduced by 79% with a few percent increase in cache access time.

1. We assume the tag bits cannot be put into a drowsy state.

Table 4.3: Run-time increase and normalized leakage power.

benchmark	run-time increase (%)		normalized leakage (%)	
	drowsy	decay	drowsy	decay
bzip2	0.77	0.38	32 (18)	33
crafty	0.46	1.08	36 (22)	103
eon	0.36	0.13	34 (20)	35
gap	0.46	0.90	37 (23)	29
gcc	0.06	0.00	32 (18)	96
gzip	2.07	1.23	37 (24)	77
mcf	0.66	0.04	36 (23)	34
parser	1.17	1.75	31 (17)	58
perl	0.87	6.97	31 (17)	198
twolf	0.87	0.29	31 (16)	34
vortex	0.25	0.78	49 (39)	112
vpr	0.89	0.84	31 (16)	70
ammp	0.54	0.12	52 (42)	73
applu	0.50	0.00	33 (19)	41
apsi	0.32	0.02	33 (19)	30
art	0.61	0.01	33 (18)	44
equake	0.03	0.04	33 (18)	10
facerec	0.15	0.03	38 (25)	22
fma3d	0.14	0.87	33 (19)	125
galgel	0.31	0.00	34 (20)	68
lucas	0.66	0.00	37 (24)	25
mesa	0.21	0.09	35 (22)	21
mgrid	0.49	0.00	33 (19)	42
sixtrack	0.34	0.18	35 (21)	52
swim	0.61	0.00	34 (20)	34
wupwise	0.20	0.01	33 (19)	34
AVG	0.57	0.64	35 (21)	55

The normalized leakage figures in the parentheses are calculated using the leakage power of the 6-transistor static memory cell implemented with 300*mV* high-V_{TH} access transistors.

Compared with the cache decay technique, the drowsy data cache shows less run-time impact as well as more leakage energy reduction on average. The drowsy data cache has relatively uniform leakage reduction over the entire SPEC2K benchmarks while the decay data cache does not. Furthermore, in some applications such as *crafty*, *fma3d*, *perl*, and *vortex* (see the shaded region in Table 4.3), the system using the cache decay technique dissipates more energy than the regular cache and it shows more run-time impact due to the relatively high number of extra L2 cache accesses incurred by the premature turn-off of live cache lines. To prevent these outlier effects, a sophisticated tuning technique on a per application is required for the cache decay technique.

In terms of extra hardware, both techniques require additional transistors to control the power mode of the cache lines. Comparing the access time of caches, the cache decay technique using gated-V_{DD} memory cells has an additional transistor in its critical path. Therefore, the access time of our drowsy cache without using high-$\mathrm{V_{TH}}$ transistors is slightly faster than that of the decay cache. In the case that high-$\mathrm{V_{TH}}$ transistors are employed to reduce further leakage power in the drowsy cache, the cache access time of both techniques will be very similar.

4.5 Chapter Summary

The relative merits of both approaches — drowsy cache and cache decay techniques — depend strongly on the latency and size of L2 caches. If the L2 cache is larger than 512K bytes, as is common in today's high performance microprocessors, the performance of the drowsy cache approach improves relative to that of the decay cache. Furthermore, in small low-power systems with no L2 cache and in which L1 misses require an off-chip access, the drowsy cache approach shows a strong advantage. During our investigations of drowsy data caches we found that our simplest policy—where cache lines are periodically put into a low-power mode without regard to their access histories—can reduce the cache's leakage power consumption by 60-75%. Our comparisons with the cache decay algorithm, which uses a state-destroying gated-V_{DD} cache to reduce leakage power, indicate that the drowsy technique not only offers better energy savings but also lacks the pathological behavior—which can actually increase power consumption—that is inherent in the gated-V_{DD} based technique.

We believe that our combination of a simple circuit technique with a simple microarchitectural mechanism provides sufficient leakage power savings at a modest performance impact to make more complex solutions unattractive. Since the cost of misprediction in a drowsy cache is low both in terms of power and performance overhead, it is especially useful for embedded processors that lack on-chip L2 caches. On a miss, a drowsy cache need only wake up the drowsy line that is already in the cache as opposed to gated-V_{DD} based designs, which would have to perform a costly off-chip access to main memory to re-load the line.

Chapter 5

Drowsy Instruction Caches

According to the experimental results presented in Chapter 4, we were able to make two observations on data caches. First, the working sets of the data caches are very small. Second, more than 60% of working sets in an update time window will not be used in next consecutive update windows. The first characteristic implies that only a small fraction of data cache lines are needed to be in active mode while the rest of the lines can be in drowsy mode to reduce the data cache leakage power. This also helps to minimize the performance loss because only a small number of extra penalty cycles are incurred by waking up the drowsy working sets.

The second one means that 40% of the working sets put into the drowsy mode will not need to be woken up soon, and that past accesses are not always a good indication of the future cache line uses. Furthermore, in the *non-blocking* data cache for the out-of-order execution core, a significant fraction of the wake-up latencies could be hidden because the other load/store instructions or other non-load/store instructions waiting in the instruction queue can be executed concurrently while waking up the drowsy cache lines; this makes the *memory impact* in Eq. 4-1 is relatively low.

In this chapter, we propose microarchitectural controls for drowsy instruction caches — as opposed to data caches that were proposed in Chapter 4. We found that while our previous algorithm was very effective for data caches, it does not work well for instruction caches because of different cache access characteristics. It is commonly known that instruction caches have more spatial and less temporal locality than data caches. Therefore, it is not surprising that a different cache control policy exploiting more spatial locality is required to reduce the instruction cache leakage power with a minimum performance loss.

5.1 Characterizing Working Set

According to our experiments, the desirable data cache access characteristics making the simple policy work well are not the same for instruction caches, because of the different access patterns. Table 5.1 shows the working set characteristics of a sub-set of the SPEC2K benchmarks when a 32KB 2-way set associative instruction cache with 1024 32-

Table 5.1: Instruction cache working set characteristics.

benchmark	working set	number of accesses	accesses / line	accesses / cycle
bzip2	3%	949.8	28.5	0.46
crafty	28%	1285.2	4.5	0.63
gcc	1%	1342.7	137.1	0.66
parser	4%	993.4	25.2	0.49
vortex	21%	994.7	4.6	0.49
ammp	3%	590.1	17.7	0.29
equake	18%	1874.6	10.0	0.92
fma3d	14%	1427.7	10.0	0.70
mgrid	13%	694.2	5.3	0.34
swim	6%	335.8	5.7	0.16
AVG	10%	1005.9	9.8	0.38

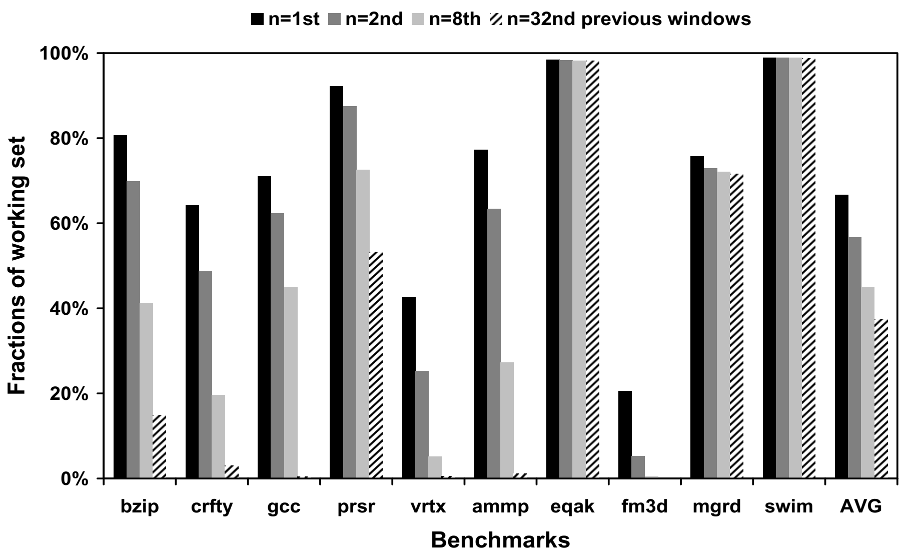
A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used.

byte lines and a 2048 cycle update window are used. The average working set percentage of the instruction cache is similar to that of the data cache, but in many workloads such as *crafty*, *vortex*, *fma3d*, and *mgrid*, the fractions of the working sets are very high. (See Table B.4 for the working set characteristics of the entire SPEC2K benchmarks.)

When a simple policy is employed to reduce the leakage power, the performance loss of those workloads will be significant because more cache lines need to be woken up in a fixed time window size. Moreover, the stalls at the front-end machine are directly translated into the performance loss unlike the non-blocking data caches; the memory impact in Eq. 4-1 for the instruction caches will be close to 1.

Figure 5.1 shows the instruction cache working set reuse characteristics — the fraction of accesses that are the same as those in the *n-th* previous window for a sub-set of SPEC2K benchmarks. As stated in the previous chapter, the results in the figure specify

Figure 5.1: Instruction cache working set reuse characteristics.



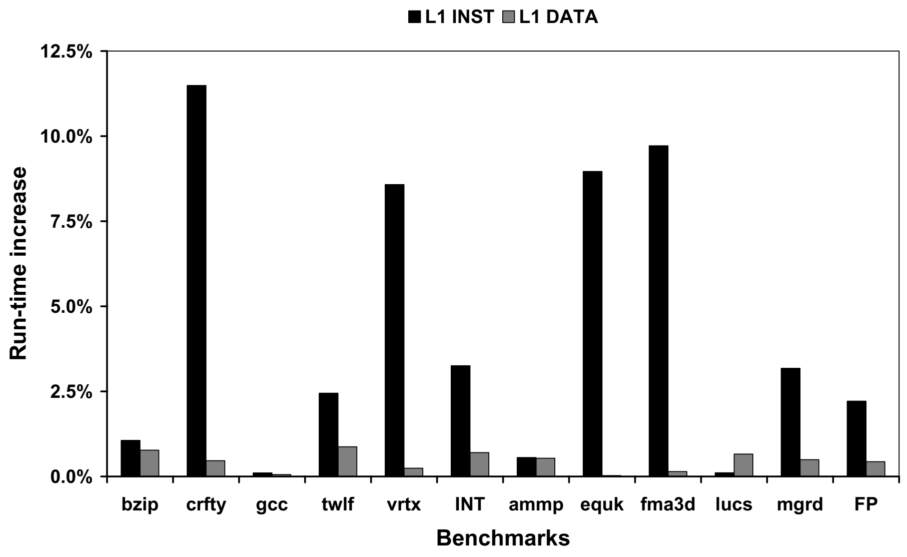
A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used. The fractions of accesses that are the same as in the n-th previous window in the current 2K cycle window size.

what fraction of references in a current window are to lines that had been accessed 1, 2, 8, or 32 windows before. The fractions of the reused working sets from earlier windows in the instruction cache are much higher than those in the data cache. In fact, the working sets of instruction caches barely change for long cycles in many workloads like *gzip*, *mcf*, *parser*, *art*, *lucas*, *equake*, and *swim*. (See Table B.5 for the statistics of the entire SPEC2K benchmarks.) In other words, the past access patterns for the instruction cache lines are good indicators for the future usage of the lines unlike data caches.

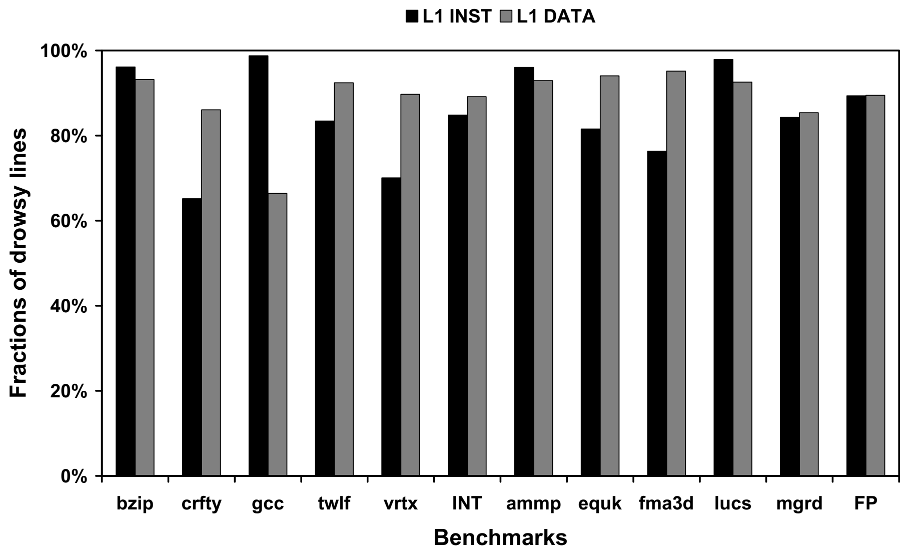
Figure 5.2 shows the run-time increase and the fraction of drowsy lines — which is proportional to leakage power reduction — of workloads using the simple policy with an 1-cycle wake-up latency and a 4K-cycle update window size, meaning that all cache lines are put into drowsy mode every 4K cycles. As seen in Figure 5.2, the worst case run-time increase is as much as 11.49% for *crafty* and the average run-time increase is 2.74% for the entire SPEC2K benchmarks even though we used an aggressive 1-cycle wake-up latency for the 32KB 2-way set associative instruction cache with 1024 32-byte lines. This is in sharp contrast with the simple policy for the data cache, where the worst run-time increase is no more than 1.2% and the average is less than 0.6% run-time increase although the fractions of drowsy lines for both caches are similar.

To investigate the memory impact by waking up cache lines in the instruction caches, we applied the working set characteristics of the 4K-cycle window size, where the average percentage of the working sets is 12.7%, to the expected *worst-case execution time increase* model from Eq. 4-1. This model gives 3.1% worst-case execution time increase when the memory impact 1 is assumed, which is very close to the actual average run-time increase — 2.7%. We confirmed that the memory impact for the instruction

Figure 5.2: Comparison of drowsy instruction and data caches.



(a) Run-time increases



(b) Fractions of drowsy lines

We use a simple policy with 4K update window size and 1-cycle drowsy-line wake-up latency — see Table B.6 for the results of all the SPEC 2K benchmarks.

cache is close to 1. This means that wake-up penalties are directly translated into the execution time increase.

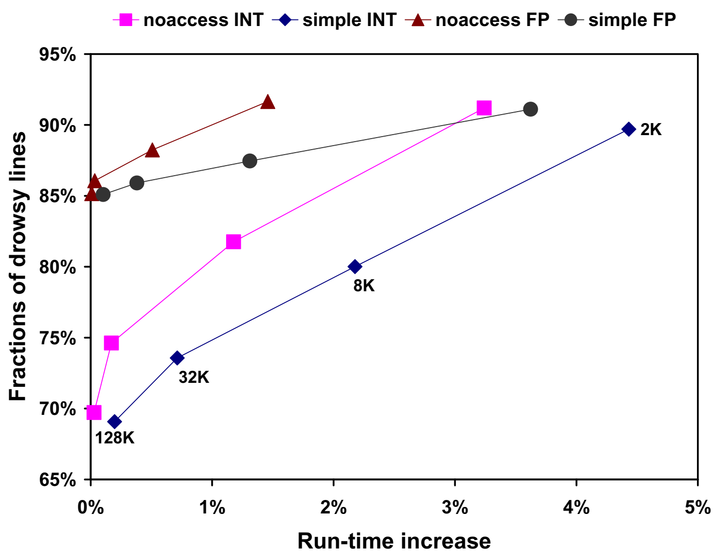
5.2 Policy Comparison

Considering the working set characteristics and the high memory impact of the instruction caches, the *noaccess* policy — putting only lines that have not been accessed in a period into drowsy mode — seems to be more promising than the *simple* policy. Figure 5.3 shows the run-time increase and the fraction of drowsy line of the noaccess and simple policies. In this experiments, we used the 128K, 32K, 8K, and 2K update window sizes and 1-cycle drowsy-line wake-up latency.

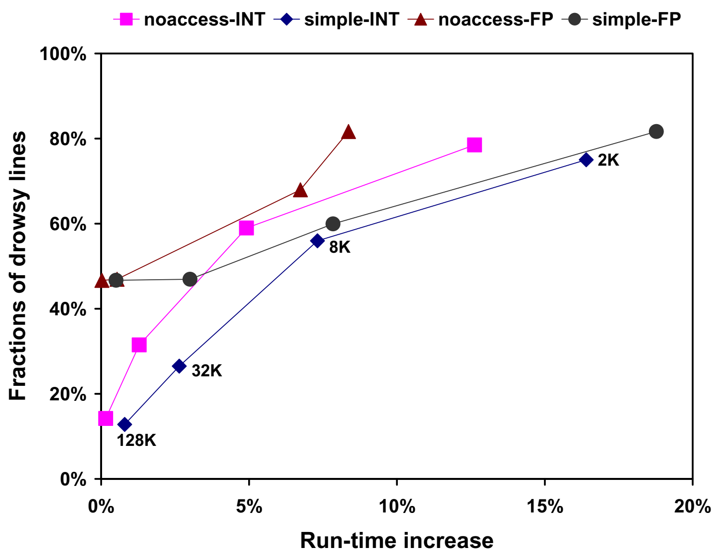
As expected, the noaccess policy performs better in terms of both the run-time impact and leakage reduction for all workloads and update windows. The noaccess policy shows 2.36%, 0.85%, 0.10%, and 0.02% average run-time increases with 91.4%, 85.3%, 80.8%, and 78.0% average fractions of drowsy lines, while the simple policy shows 4.03%, 1.75%, 0.55%, and 0.15% average run-time increases with 90.46%, 84.02%, 80.21%, and 77.70% average fractions of drowsy lines for 2K, 8K, 32K, and 128K update window sizes. Compared to the simple policy, the noaccess policy reduces the run-time increases by 41.36%, 51.63%, 81.27%, 87.94% with a slight decrease in fractions of drowsy lines for 2K, 8K, 32K, and 128K update window sizes.

Comparing the worst case run-time increase and fraction of drowsy lines among the entire SPEC2K benchmark programs, the simple policy shows 18.8%, 7.8%, 3.0%, and 0.8% run-time increases with 75.0%, 56.0%, 26.5%, and 2.8% fractions of drowsy lines while the noaccess policy shows 12.6%, 6.7%, 1.3%, and 0.2% run-time increases

Figure 5.3: Comparison of noaccess and simple policies.



(a) SPEC2K benchmark average



(b) SPEC2K benchmark worst case

We compare noaccess and simple policies with 128K, 32K, 8K, and 2K update window sizes and 1-cycle drowsy-line wake-up latency.

with 78.5%, 59.0%, 31.5%, and 14.2% fractions of drowsy lines for 2K, 8K, 32K, and 128K update window sizes.

Table 5.2 shows the average, best, and worst run-time increases and fractions of drowsy line comparison of noaccess and simple policies. The numbers in the left and right sides represent the results for the noaccess and simple policies, respectively. In almost all cases, the noaccess policy for the instruction caches outperforms the simple policy while the noaccess policy in data caches shows smaller fractions of drowsy lines than the simple policy.

Table 5.2: Comparison of noaccess and simple policies.

		run-tim increases (%)			
		2K	**8K**	**32K**	**128K**
AVG	**INT**	3.24 / 4.43	1.18 / 2.18	0.17 / 0.71	0.03 / 0.20
	FP	1.46 / 3.62	0.51 / 1.31	0.03 / 0.38	0.01 / 0.10
BEST	**INT**	0.02 / 0.05	0.02 / 0.01	0.00 / 0.01	0.00 / 0.01
	FP	0.00 / 0.08	0.00 / 0.01	0.00 / 0.01	0.00 / 0.00
WORST	**INT**	12.63 / 16.40	4.92 / 7.31	1.29 / 2.64	0.16 / 0.80
	FP	8.36 / 18.77	6.74 / 7.84	0.54 / 3.00	0.03 / 0.05

		fractions of drowsy lines (%)			
		2K	**8K**	**32K**	**128K**
AVG	**INT**	91.2 / 89.7	81.8 / 80.0	74.6 / 73.6	69.7 / 69.0
	FP	91.7 / 91.1	88.2 / 87.5	86.0 / 85.9	85.2 / 85.1
BEST	**INT**	99.4 / 99.4	99.3 / 99.2	98.9 / 99.8	98.6 / 98.6
	FP	99.5 / 99.5	99.4 / 99.4	99.3 / 99.3	99.2 / 99.3
WORST	**INT**	78.5 / 75.0	59.00 / 56.00	31.5 / 26.5	14.2 / 12.8
	FP	81.7 / 81.7	68.00 / 59.9	46.9 / 46.9	46.7 / 46.7

A 32KB, 2-way set associative cache with 1024 32-byte lines is used. The numbers in the left and right sides represent the results for noaccess and simple policies, respectively.

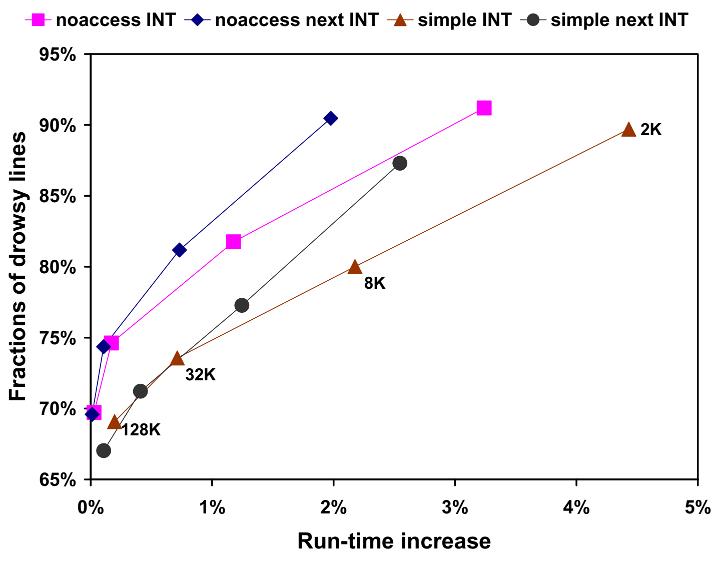
In the noaccess policy, a per-line access history counter is required. This is expensive: implementing a counter for 4K or 8K cycles requires 12 or 13 extra bits per cache line. However, the counter can be approximately implemented in a hierarchical way — one global counter with one two-bit register per cache line. To implement a counter counting 4K cycles, the global counter asserts a signal, fed to 2-bit local counters in all the cache lines, every 512 cycles. Again, the periodic signal from the global counter increases the counters in the cache line. Whenever there is any access to a cache line, the counter in the line is asynchronously reset to “0.” As soon as the counter is set to “3,” implying the line has not been accessed for 4K cycles, the line is put into drowsy mode.

5.3 Next Line Early Wake-up

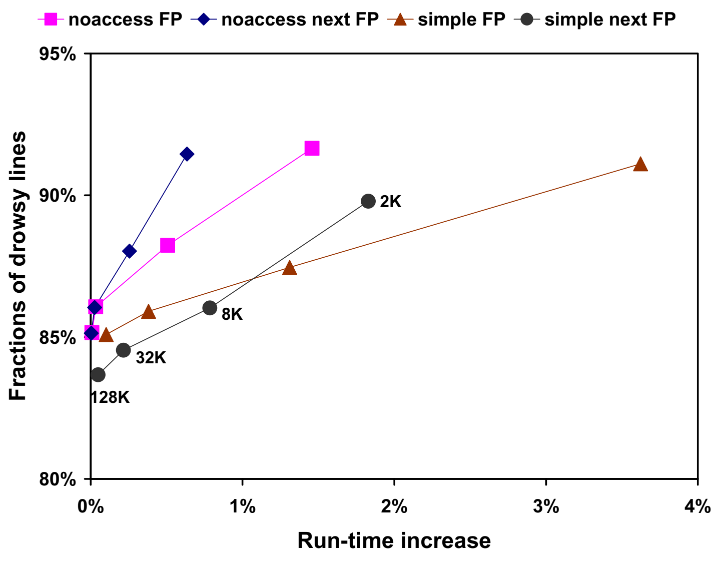
Generally, instruction caches access cache lines sequentially except when branches are encountered. In many previous studies, the next-line prefetch has been shown effective for improving cache performance [62, 63, 64, 65, 66]. To reduce the performance loss by wake-up penalties, the same concept can be applied to drowsy cache lines. As soon as a cache line in row *n* that has been in drowsy mode is woken up, the wake-up signal from the currently woken up cache line awakens the adjacent cache line in row *n+1*. The implementation of the next line early wake-up requires only a slight modification of the existing drowsy cache line design shown in Figure 4.5: the final decoder signal in row *n* is also connected to the *wakeup* node of the drowsy flip-flop in row *n+1*.

Figure 5.4 shows the run-time increase and the fraction of drowsy lines of the noaccess and simple policies with the early wake-ups. In this experiment, we also show the noaccess and simple policies without early wake-up to see whether the simple policy

Figure 5.4: Comparisons of noaccess and simple next line early wake-up policies.



(a) SPEC2K integer benchmarks



(b) SPEC2K floating-point benchmarks

We compare noaccess and simple policies with 128K, 32K, 8K, and 2K update window sizes and 1-cycle drowsy-line wake-up latency.

with early wake-up performs better than the noaccess policy without early wake-up. As seen in Figure 5.4, the early wake-up reduces the run-time increases significantly with only a slight decrease in the fractions of drowsy lines.

Compared to the noaccess policy without the early wake-up, the noaccess with the early wake-up reduces the run-time increases by 44.4%, 41.4%, 35.6%, and 55.5%, respectively with negligible decreases in the fractions of drowsy lines for 2K, 8K, 32K, and 128K update window sizes. In addition, the simple policy with the early wake-up reduces the run-time increases by 42.5%, 42.8%, 42.5%, and 45.4%, respectively, compared to the simple policy without early wake-up.

Compared to the simple policy with early wake-up, the noaccess policy without early wake-up reduces the run-time increases by 16.9%, 67.3%, and 76.9% with 2~4% more drowsy lines for 8K, 32K, and 128K update window sizes; the simple policy with early wake-up outperforms the noaccess policy without early wake-up by 7.9% only in the 2K update window size.

In conclusion, the noaccess policy performs better than the simple one due to higher working set re-use. Moreover, the next line early wake-up — the simplest form of the prediction for next-accessed cache line — reduces the run-time increases significantly without compromising the leakage savings. However, the fractions of drowsy lines in the instruction caches are less than those of data caches. Therefore, to further reduce the leakage power, the leakage through the bit-lines and access transistors should also be reduced.

Unlike data caches, the use of high-V_{TH} transistors for the access transistors in the 6-transistor memory cell may not be a good solution for instruction caches, because the instruction cache access or cycle time is critical in determining the cycle-time of the processor; this is one of the reasons why direct mapped caches have been preferred to set-associative caches.

In the following sections, we propose a gated bit-line precharge (or bit-line isolation) technique to further reduce the leakage power of the instruction caches. This technique reduces the leakage power through the bit-lines and access transistors. To minimize the performance overhead, we also propose a next sub-bank predictor that allows processors to precharge the predicted next sub-bank in advance.

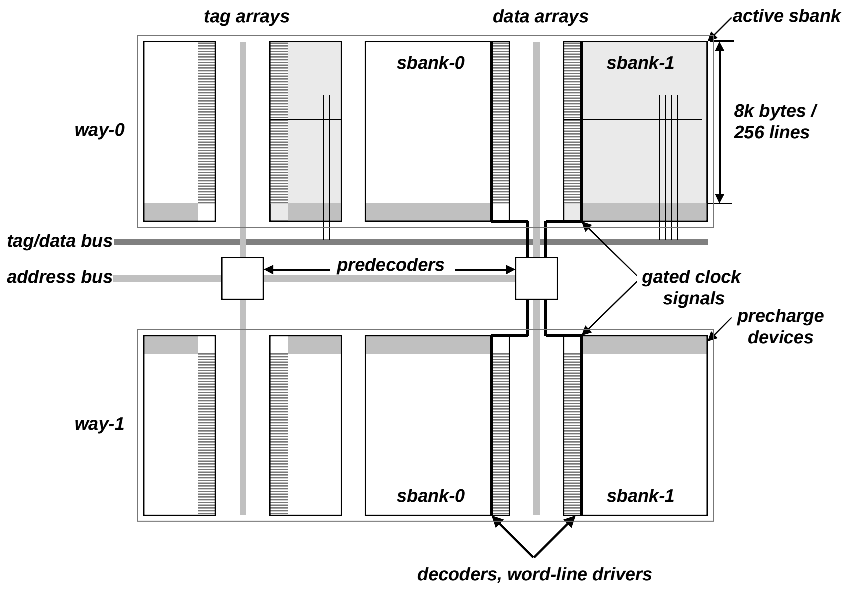
5.4 Gated Bit-Line Precharge

5.4.1 On-Demand Gated Bit-Line Precharge

Several researchers have proposed the use of sub-banks as a means of reducing dynamic power consumption in caches. In [5], a cache is partitioned into several sub-banks (or sub-arrays), and on each access only a limited set of sub-banks are checked for their contents. This approach reduces the dynamic power consumption at the cost of slightly increasing the cache access time due to the additional decoder logic needed to index the sub-banks.

In [39], a gated bit-line precharge technique is applied to sub-banks to reduce the leakage power by the bit-lines. It cuts off the leakage paths from the supply voltage to the bit-lines by turning off the precharge circuits; see Figure 3.1 for the leakage path. Figure 5.5 shows a 32KB, 2-way set associative sub-banked cache organization supporting on-demand gated bit-line precharge. The cache consists of 4 8KB sub-banks and the precharge clock signal is applied to only one active sub-bank through the signal gating cir-

Figure 5.5: Cache organization supporting on-demand gated precharge.



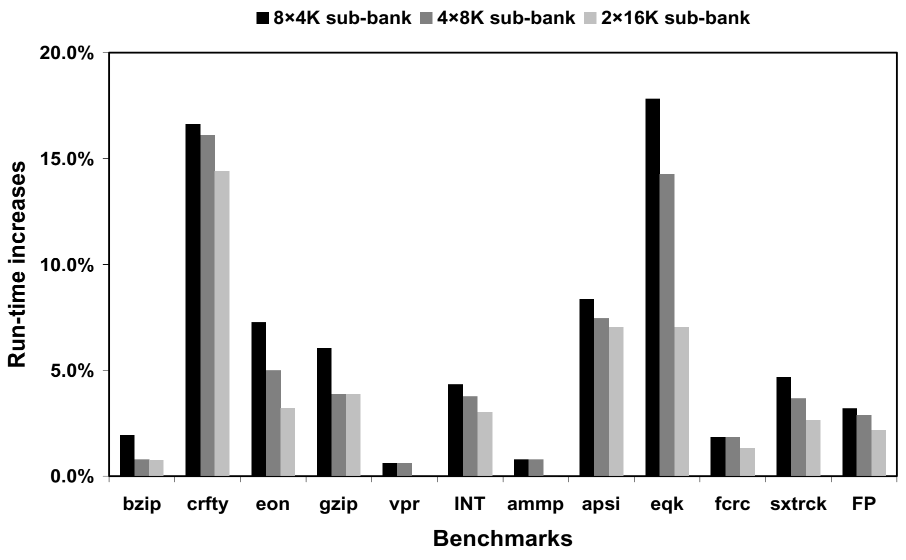
A 32KB, 2-way set associative instruction cache of 4×8KB sub-banks is assumed.

cuit. Hence, only the bit-lines in the active sub-bank are precharged, and those in inactive sub-banks are floated by the gated precharge circuits.

However, when a processor leaves the currently accessing sub-bank and initiates an access to a different sub-bank that has been in the inactive mode it should wait at least one cycle for the bit-lines in the sub-bank to be fully precharged. Since the address decoding and the bit-line precharging should be done concurrently and the sub-bank index can be identified only after the address decoding, a processor performance penalty is incurred; we call this technique *on-demand* gated bit-line precharge.

Figure 5.6 shows the run-time increases of a 32KB, 2-way set associative instruction cache of 8×4KB, 4×8KB, and 2×16KB sub-banks with an 1-cycle penalty to initiate the precharge of a newly accessed sub-bank that has been in the inactive mode. According to the results, the use of on-demand gated bit-line precharge can result in the 23.3%,

Figure 5.6: Run-time increases of on-demand gated precharge.



A 32KB, 2-way set associative instruction cache and 8×4KB, 4×8KB, and 2×16KB sub-banks are used with a 1-cycle penalty to initiate a new precharge of a sub-bank.

21.7%, and 19.1% (*fma3d*) worst case run-time increases. The average performance losses are 3.8%, 3.3%, and 2.6%, for 8×4KB, 4×8KB, and 2×16KB sub-bank, 32KB 2-way set associative instruction caches, respectively.

As the size of the sub-bank increases, the performance loss decreases because a larger sub-bank can cover more cache accesses without transitions to another sub-bank. In some workloads, the run-time increases is zero or approaches zero due to small size of the code foot-print — all the executed code fits into one or two sub-banks, and some performance penalties are hidden by the cache misses. (See Table B.7 for the entire SPEC2K benchmark results.)

5.4.2 Characterization of Sub-Bank Transitions

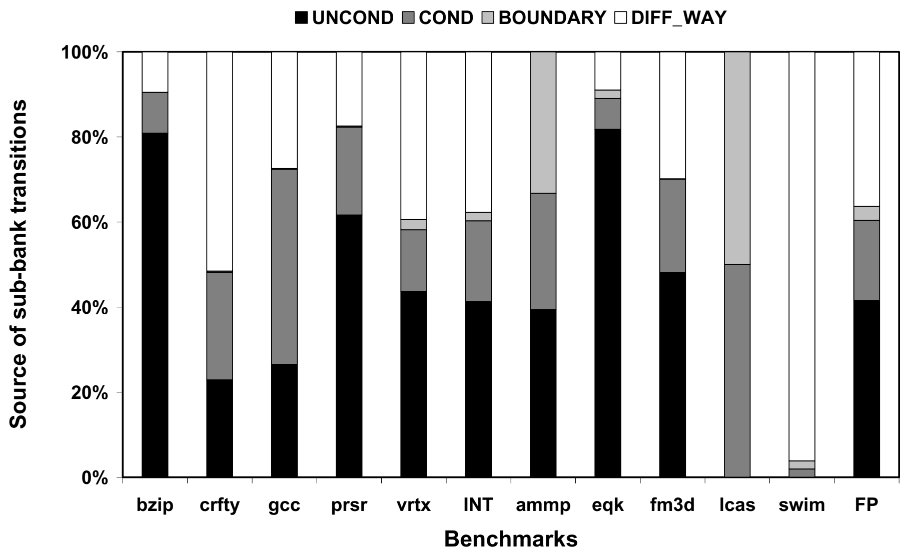
The high run-time increase of some workloads with the on-demand gated bit-line precharge is unacceptable for the high-performance processors. These performance losses are incurred when cache accesses transit from one sub-bank to another. However, we can predictively initiate precharges of the next sub-bank one cycle ahead to eliminate this performance loss.

To design the predictor, we need to identify the sources of these sub-bank transitions. The fundamental insight is that the transitions between sub-banks are often correlated with the specific types of instructions. For example, the program counter, which is the instruction cache access index or pointer, remains in small cache regions for relatively long periods of cycles as a result of loops. On the other hand, there are often abrupt changes of the cache access pointer when subroutines are called or subroutines return to the caller routines. On the other hands, most conditional branches stay within the current cache region and it is less common that these branches jump across page boundaries.

Figure 5.7 shows the sources of sub-bank transitions. “UCOND”, “COND”, “BOUNDARY”, and “DIFF_WAY” represent the sub-bank transitions from unconditional and conditional — including direct and indirect — jumps, by sequential accesses between two adjacent sub-bank boundaries, and by sequential accesses of next sets in different ways, respectively. It is obvious why the jumps in the first two categories — “UNCOND” and “COND” incur the sub-bank transitions. However, “BOUNDARY” and “DIFF_WAY” needed more explanation.

When the cache access pointer reaches the last line of one sub-bank (e.g., row “255” in the 8KB sub-bank “0”), the next sequential access will be to the first line of the other sub-bank (e.g., line “0” in the sub-bank “1”). This is an example of the “BOUNDARY” case. In “DIFF_WAY”, a different sub-bank is accessed during a sequential access of the program code. This happens when the sequence of code lines are not loaded into a

Figure 5.7: Source of bank transitions.



A 32KB, 2-way set associative instruction cache of 8×4KB sub-banks with a perfect branch predictor is used.

sub-bank sequentially but spread across different cache ways. For example, the code was originally loaded sequentially in the cache lines, but some lines were replaced and then reloaded into different ways having the same cache set indices.

In both SPEC2K integer and floating-point workloads, the unconditional jumps are responsible for 40% of the sub-bank transitions in averages. The sequential accesses of the next sets in different ways also cause 40% of the transitions. However, *lucas*, in which the size of the code foot-print is very small, is different. The conditional jumps and the sequential accesses between two sub-bank boundaries are the dominant sub-bank transitions, because *lucas* keeps looping between two sub-bank boundaries. In *swim* execution stays in one sub-bank for 95% of the execution cycles, and the sequential accesses of a next set in a different way are responsible for most of the sub-bank transitions.

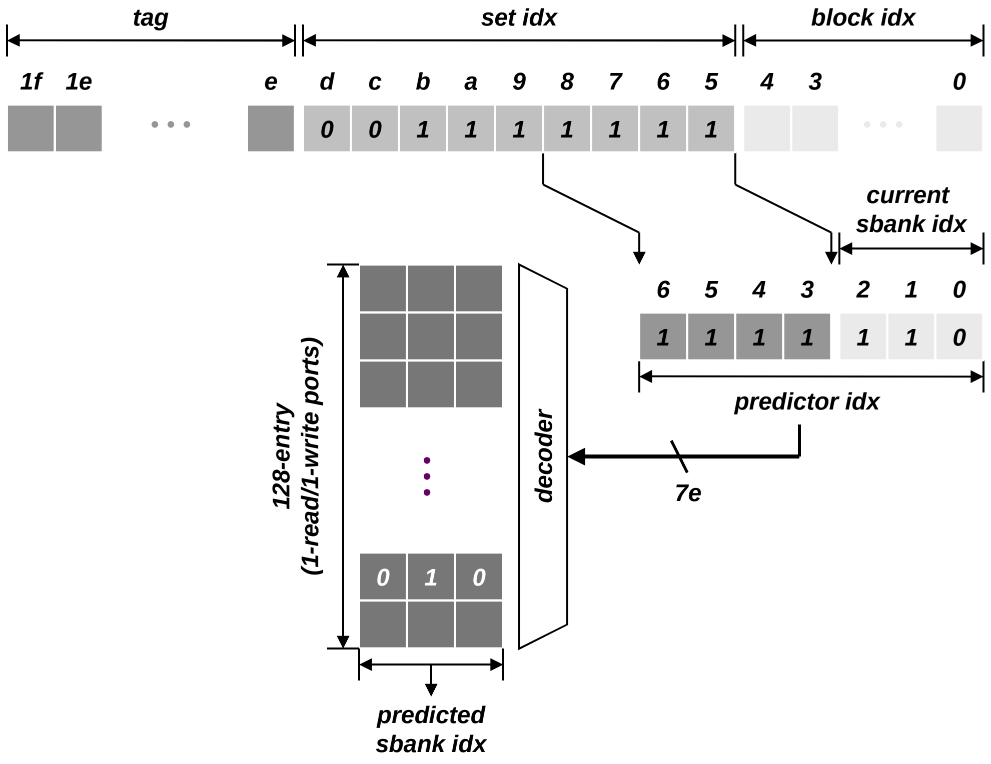
5.4.3 Predictive Gated Bit-Line Precharge

According to our analyses on sub-bank transitions, a significant fraction of them are triggered by unconditional jumps caused by sub-routine calls and returns, or by accessing sets in different ways during sequential accesses of cache lines. Therefore, these transition points (or addresses) from one sub-bank to another in the instruction caches are quite predictable.

The basic idea of the next target sub-bank predictor is as follows. When an instruction triggering the transitions is encountered (e.g., from the sub-bank `0x6` to `0x2`), a next target sub-bank predictor index (e.g., `0x7e`) is generated from the cache set index of the previous cycle access (e.g., `0x7f`) and the sub-bank index of the current cycle one (e.g., `0x6`). With the predictor index, the target sub-bank index (e.g., `0x2`) is stored in the predictor. (See Figure 5.8 for how to generate the predictor index.) When the cache set of `0x7f` in the sub-bank `0x6` is accessed in later cycles, the pointer gives the next target sub-bank index — `0x2`. Since the instructions cache equipped with a single port only allows the processor to access a single cache line in a cycle, the set address instead of the individual instruction address is enough to detect the sub-bank transitions and to index predictors.

Figure 5.9 shows the run-time increase vs. the predictor accuracy for a sub-set of SPEC2K integer and floating-point workloads. These workloads are selected for the illustration because they showed poor performance while the rest of the workloads showed negligible run-time increases. We used a 32KB, 2-way set associative instruction cache of 8×4KB sub-banks with 64-, 128-, 256-, 512-, and 1K-entry predictors. As we increase the

Figure 5.8: Next sub-bank predictor index generation.

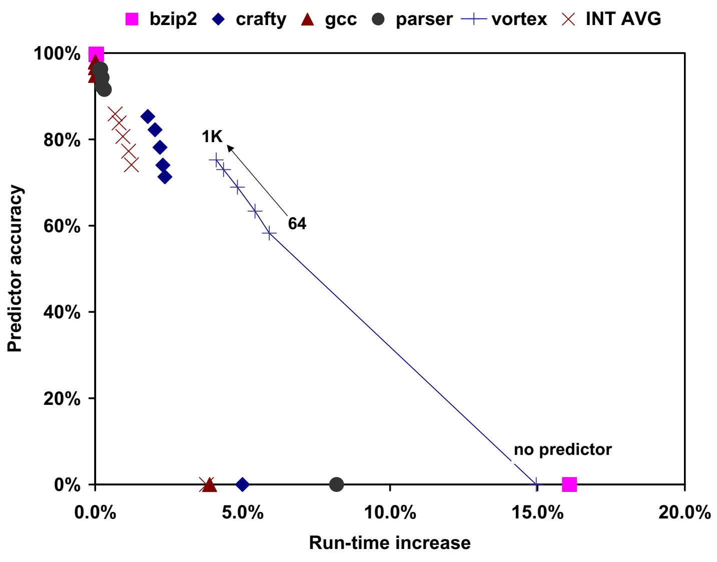


A 32-bit address space with a 512-set and 32-byte block size cache is assumed. The number in the square represents the bit position index.

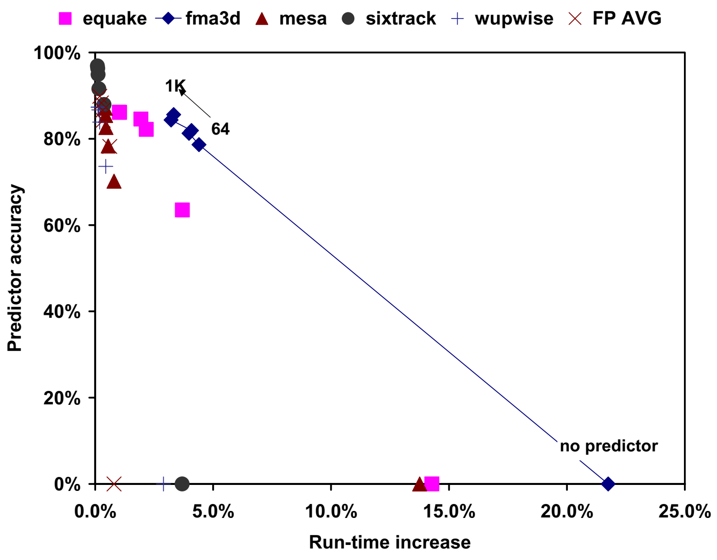
number of the predictor entries, the average accuracy increases by 75.5%, 79.7%, 82.7%, 85.4%, and 87.3% with the 3.3%, 0.93%, 0.74%, 0.63%, 0.54%, and 0.44% average run-time increases for 64-, 128-, 256-, 512-, and 1K-entry predictors, respectively. Compared to the run-time increase without the predictors — on-demand gated bit-line precharge, the proposed predictor reduces the run-time increases by 72%, 78%, 81%, 84%, and 87% for 64-, 128-, 256-, 512-, and 1K-entry predictors, respectively.

With a 64-entry predictor, which was the smallest size predictor we configured, we could reduce the run-time increases of *equake, fma3d, mesa, bzip2, crafty, parser,* and *vortex* from 14.3%, 21.7%, 13.8%, 16.1%, 5.0%, 8.2%, and 15.0% to 3.70%, 4.40%, 0.8%, 0.03%, 2.36%, 0.31%, and 5.90%, respectively. For those workloads, we reduce the run-

Figure 5.9: Run-time increase vs. predictor accuracy.



(a) SPEC2K integer benchmarks



(b) SPEC2K floating-point benchmarks

A 32KB, 2-way set associative instruction cache and 8×4KB sub-banks with 64-, 128-, 256-, 512-, and 1K-entry predictors are used.

time increases by 74%, 80%, 94%, ~100%, 53%, 96%, and 61%, respectively. These results prove that a small size predictor is also very effective at reducing the performance loss of the gated bit-line precharge.

Theoretically, this technique reduces the bit-line leakage power by 87.5% in the 8×4KB sub-bank instruction cache, because only one sub-bank is precharged among 8 sub-banks. However, it takes a finite time until the floated bit-line conditions are balanced [39, 41]. Therefore, the actual bit-line leakage saving is less than 87.5%.

In our previously proposed technique [76, 77], we used a *content addressable memory* (CAM), which is very expensive, to store and match the full address bits of instructions for the next target sub-bank predictor. However, it turns out that we can achieve a similar accuracy with much simpler structure requiring less hardware.

When a sub-bank transition occurs by accessing a different way of a set during the sequential access of the instruction cache, the cache access pointer returns from that sub-bank to the next sequential cache set in the original sub-bank. In this case, the cache access pointer stays in the different way (or sub-bank) for at most a couple of cycles before returning to the original sub-bank.

In addition, it is not uncommon for the cache access pointer to leave the current sub-bank and come back to it in a few cycles due to conditional branch misprediction recovery. However, turning off the precharge circuits of the original sub-bank and turning them on again a few cycles later is detrimental to both performance and energy saving. Furthermore, all those transient cases stress the next target sub-bank predictors.

To reduce the performance loss for the above two cases, we can turn off a sub-bank a few more cycle later even though the cache access pointer leaves the sub-bank or the

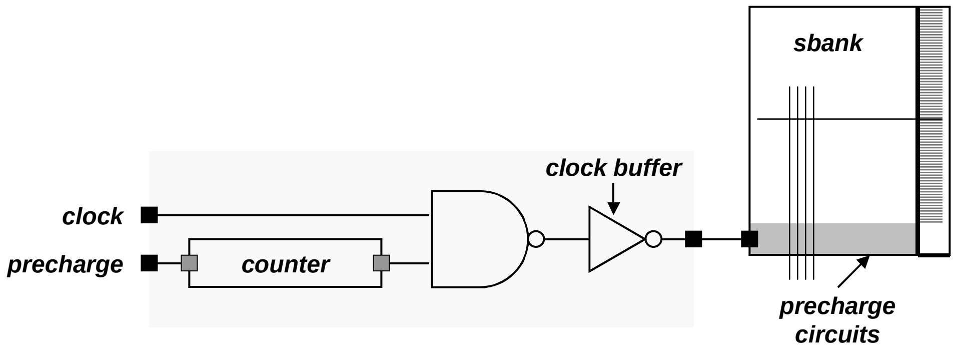


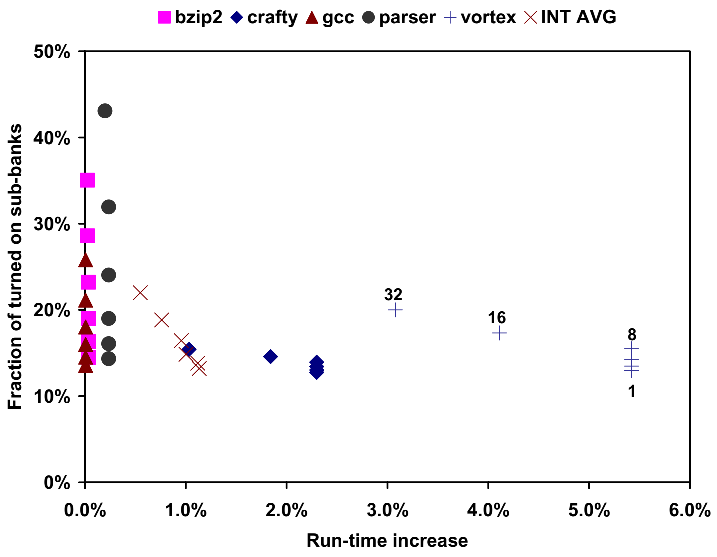
Figure 5.10: Gated precharge clock circuit with a decay counter.

precharge signal is turned off. This can be easily implemented by adding a counter to the gated pre-charge circuits as shown in Figure 5.10. As soon as the precharge signal is turned off, the counter starts to decrease. When the counter reaches “0,” the precharge clock signal is gated and turned off. The sub-bank precharge circuits are turned on until the counter value becomes “0”.

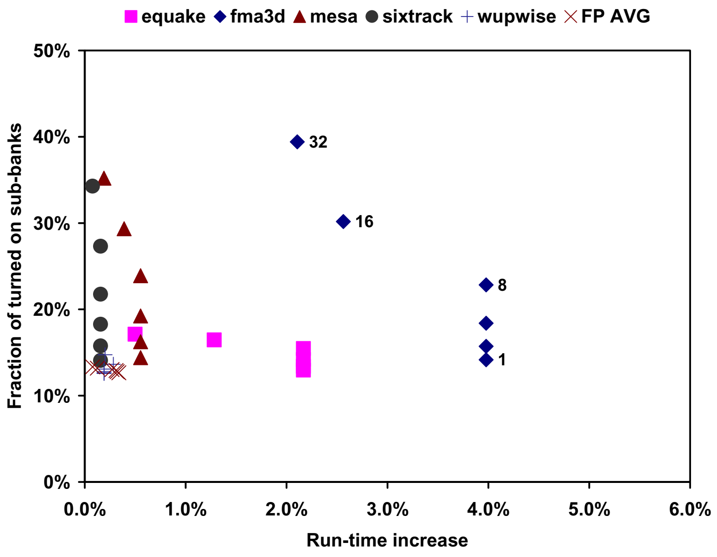
As we increase the counter interval or sub-bank turn-off delay period, we will be able to cut more performance loss, because, this allows several sub-banks to be turned on for a short period of time. We have more chance of transition to a turned-on sub-bank. However, we will get less leakage savings, because the saving is inversely proportional to the number of turned-on sub-banks.

Figure 5.11 shows the run-time increase vs. the fractions of turned-on sub-banks for the sub-set of SPEC2K integer and floating-point workloads shown in Figure 5.9. In this experiment, we increased the turning-off delay period by 1, 2, 4, 8, 16, and 32 cycles with the same cache configuration used in Figure 5.9. As we increase the delay period, the average fractions of turned-on sub-banks increases are 13.2%, 13.8%, 14.7%, 16.0%, and 17.6%, and the run-time decreases are 0.74%, 0.72%, 0.66%, 0.62%, 0.45%, and 0.32% for the 1-, 2-, 4-, 8-, 16-, and 32-cycle counter intervals with a 128-entry predictor. Com-

Figure 5.11: Run-time increase vs. fraction of turned on sub-banks.



(a) SPEC2K integer benchmarks



(b) SPEC2K floating-point benchmarks

A 32KB, 2-way set associative instruction cache and 8×4KB sub-banks with 64-, 128-, 256-, 512-, and 1K-entry predictors are used.

pared to the run-time increase without the turn-off delay period, the proposed technique reduces the average run-time increases by 2.11%, 10.90%, 16.73%, 39.73%, and 57.35%, respectively with at most a 4.4% increase in the fraction of turn-on sub-banks for the 2-, 4-, 8-, 16-, and 32-cycle counter intervals.

In general, as we increase the turn-off delay period or counter interval, we can cut more performance loss while increasing the fractions of turned-on sub-banks. Also, seeing the run-time decrease trends, the counter interval should be more than 8 cycles to be effective at reducing the performance loss of those workloads showing poor run-time behavior. This implies that the poor performance of the predictor for some workloads is caused by conditional branches, and that the number of cycles to resolve the conditional branches is more than 8 cycles.

5.5 Run-time Impact and Leakage Power Reduction

Table 5.3 shows the run-time increase and the normalized leakage power of the proposed techniques — the noaccess drowsy policy with the early wake-up and with gated bit-line precharge. A 32KB, 2-way set associative instruction cache and 8 4KB sub-banks with a 32K-cycle update window size are used for the noaccess policy, and a 1K-entry predictor, and a 16-cycle turn-off delay period for the gated bit-line precharge is used.

On average, the noaccess policy with the 32K-cycle update window shows only a 0.07% run-time increase with the 60% leakage power reduction. In the worst case, *vortex* shows the 0.85% performance loss with the 21% leakage power saving. When we employ the gated bit-line precharge as well as the noaccess policy, the average run-time increase is

Table 5.3: Run-time increase and normalized leakage power.

benchmark	run-time increase (%)		normalized leakage (%)	
	noaccess	w/ GBP	noaccess	w/ GBP
bzip2	0.05	0.04	31	17
crafty	0.62	2.01	72	61
eon	0.06	2.79	53	43
gap	0.03	0.62	53	41
gcc	0.03	0.03	30	16
gzip	0.04	0.11	30	17
mcf	0.00	0.01	28	14
parser	0.01	0.09	31	17
perl	0.16	1.55	64	53
twolf	0.11	0.19	50	36
vortex	0.85	3.84	79	69
vpr	0.01	0.02	31	16
ammp	0.07	0.07	34	19
applu	0.01	0.01	33	19
apsi	0.11	1.32	43	30
art	0.00	0.01	28	13
equake	0.00	1.10	41	30
facerec	0.01	0.17	30	16
fma3d	0.53	2.78	66	56
galgel	0.00	0.00	28	13
lucas	0.00	0.00	29	14
mesa	0.04	0.24	42	30
mgrid	0.00	0.00	41	27
sixtrack	0.01	0.11	37	24
swim	0.01	0.00	32	17
wupwise	0.01	0.17	40	26
AVG	0.07	0.38	41	28

GBP stands for gated bit-line precharge. A 32KB, 2-way set associative instruction cache of 8×4KB sub-banks with a 1K-entry predictor and a 16-cycle gated bit-line precharge turn-off delay are used.

0.38% — less than 0.5% performance loss — while reducing 72% of the leakage power. In the worst case, *vortex* shows a 3.84% performance loss with a 31% leakage saving.

In particular, *bzip2*, *gcc*, *mcf*, *parser*, *vpr*, *applu*, *art*, *galgel*, *lucas*, and *swim* — nearly the half of the entire SPEC2K workloads — show negligible or no performance losses while achieving more than 80% leakage power reduction for the combined technique. In terms of the area overhead, the 1K-entry next target sub-bank predictor requires less than 1.2% the number of bits of the 32KB, 2-way set associative instruction cache.

Finally, we adopted the same cache line control policy for our drowsy instruction caches as the cache decay technique. Therefore, in terms of the run-time increase, our proposed state-preserving drowsy technique is much better at the same update window size because of the reduced penalty. Even though the cache decay technique works well in terms of the leakage reduction, it may not be suitable for the instruction caches. It increases the cache access time because the power gating circuitry is in the critical path of the SRAM read. The access time of instruction caches is more critical than that of data caches in determining the cycle time of the processor and translates directly into processor performance. But our proposed technique does not increase the access time of the cache.

5.6 Chapter Summary

First, we demonstrated that the application of the simple policy used for data caches did not work well for instruction caches due to different cache access patterns. According to our analysis on the instruction cache working sets, the previous access history of cache lines was a good indicator for the future usage of the lines. We compared the run-time increases and fractions of drowsy lines for both noaccess and simple policies.

The results show that noaccess policy is better in terms of both run-time increases and fractions of drowsy lines.

Second, we proposed to use a next line early wake-up technique to further reduce the performance loss, based on the insight that a significant fraction of instruction cache accesses are sequential. This scheme reduces the performance loss by more than 40% compared to the noaccess policy without the early wake-up.

Third, to further reduce the leakage power from the bit-lines, we proposed using a gated bit-line precharge technique. However, the on-demand gated bit-line precharge increases the run-time increase up to 21%. To reduce the performance loss, we proposed a next sub-bank predictor which reduced the average run-time increase by more than 70% with 64 entries. Furthermore, there is a larger design space to be explored that may improve the accuracy of the predictor.

Finally, the combination of the noaccess policy with early wake-up and gated bit-line precharge reduces the leakage power by 72% with less than 0.5% performance loss. The extra resources needed by the combined techniques are around 2% of the 32KB 2-way set associative cache area.

Chapter 6
Leakage Power Optimization for Multi-Level Caches

Traditionally, only two V_{TH}'s have been available in high-performance semiconductor process technologies, allowing cache designers limited flexibility to suppress leakage current. Thus, to reduce leakage, the focus has been on dynamic circuit and microarchitectural techniques similar to those we have explored in the previous chapters.

However, due to the increasing importance of sub-threshold leakage current, the number of available V_{TH}’s in future semiconductor process technologies will increase. Next generation 65*nm* processes are expected to support 3 V_{TH}'s and future processes are likely to provide designers with even more V_{TH} choices. This increase provides new flexibility for leakage power reduction methods, allowing new trade-offs between the V_{TH} of different parts of a cache and between different levels in the cache hierarchy. The availability of additional V_{TH}'s suggests a new examination of the trade-off between cache size and V_{TH} to reduce sub-threshold leakage power loss.

In this chapter, we present optimization techniques to reduce the leakage power of on-chip caches assuming that there are multiple V_{TH}’s available. First, we show a cache leakage optimization technique that examines the trade-off between access time and leakage power by assigning distinct V_{TH}'s to each of the four main cache components — address bus drivers, data bus drivers, decoders, and SRAM cell arrays with sense-amplifiers. Second, we show optimization techniques to reduce the leakage power of L1 and L2 on-chip caches without affecting the average memory access time.

The key results are: 1) 2 high V_{TH}'s are enough to minimize leakage in a single cache[1]; 2) if the L1 cache size is fixed, increasing the L2 size can result in much lower leakage without reducing average memory access time; 3) if the L2 size is fixed, reducing L1 size can result in lower leakage without loss of the average memory access time; and 4) smaller L1 and larger L2 caches than are typical in today’s processors result in significant leakage and dynamic power reduction without affecting the average memory access time.

6.1 Concept

In general, L1 caches are more frequently accessed than L2 caches. This implies that the overall microprocessor memory system performance is more affected by the access time of L1 caches. In contrast, the memory system performance impact from the access time of L2 caches is relatively small compared to that by the L1 caches. Furthermore, the fraction of leakage power in total cache power becomes more significant than that of dynamic power, because the leakage power is constantly consumed in the L2 caches; dynamic power of an L2 cache is dissipated only when there is an L1 cache miss initiating the L2 cache access.

This observation suggests a new approach for optimizing the leakage power of cache memory systems while minimizing the access time impact. To optimize the leakage power of the cache memory system under access time constraints, we present systematic approaches to V_{TH} assignment and cache memory hierarchy configuration. Our study is limited to hierarchies consisting of L1, L2 caches, and main memory. However, our

1. A 3rd V_{TH} is assumed for the processor.

approach is readily extended to systems with more cache levels.

First, we examine the optimization of the leakage power of individual on-chip cache memories that can be achieved if more than one V_{TH} is used. We show how many independent V_{TH}'s are needed for effective leakage power reduction and how much V_{TH} can be increased with a minimal increase of cache access time. Second, we show that L1 and L2 cache miss characteristics under SPEC2K workloads allow us to reduce total leakage as well as total dynamic power dissipation while maintaining the same overall average memory access time in the cache memory system.

6.2 Circuit Modeling Methodology

To examine trade-offs between leakage power and access time of a microprocessor cache memory system, we need cache access time and leakage power models. Rather than starting from the scratch, we could have built on a widely used cache memory model called "CACTI" [61]. This model estimates access time, dynamic energy dissipation, and area of caches for given cache configuration parameters such as total size, line size, associativity and number of ports.

However, it is based on an outdated $0.8\mu m$ CMOS technology and it applies linear scaling to obtain the figures for smaller process technologies. Also, it does not support access time and leakage power models for multiple V_{TH}'s. To address these shortcomings, we designed caches with the 70*nm* Berkeley predictive technology [4], in anticipation of the next generation of process technology, and derived our leakage power and access time models based on HSPCE simulations.

Table 6.1: Cache organizations for each cache size.

cache size (KB)	# of sub-banks	sub-bank size (KB)	sub-bank organization	
			bit-lines	word-lines
16	4	4	256	128
32	8			
64	4	16	512	256
128	8			
256	4	64	1024	512
512	8			
1024	16			

Caches were designed with sizes ranging from 16KB to 1024KB. The bit-lines and word-lines were segmented to improve access time, and sub-banks were employed to reduce dynamic power dissipation [5] as well. (See Table 6.1 for the cache sub-bank configuration.) The caches were broken into four components for the purposes of assigning distinct V_{TH}'s: *address bus drivers*, *data bus drivers*, *decoders*, and *6T-SRAM cell arrays* with *sense-amplifiers*. Figure 6.1 illustrates the cache sub-bank organization used in this study. For simplicity, we assume that one distinct V_{TH} can be assigned to each component.

The circuit styles and the “W/L” ratios of transistors for the circuits are the same as those in the CACTI model but optimized for the 70*nm* technology. For address and data bus interconnects, we employed an “H-tree” topology and inserted repeaters on each branch of the buses to optimize the access time of the caches. The interconnect capacitance and resistance for the long wires such as bit-lines, word-lines, address, and data bus wires are estimated by the predictor in [4].

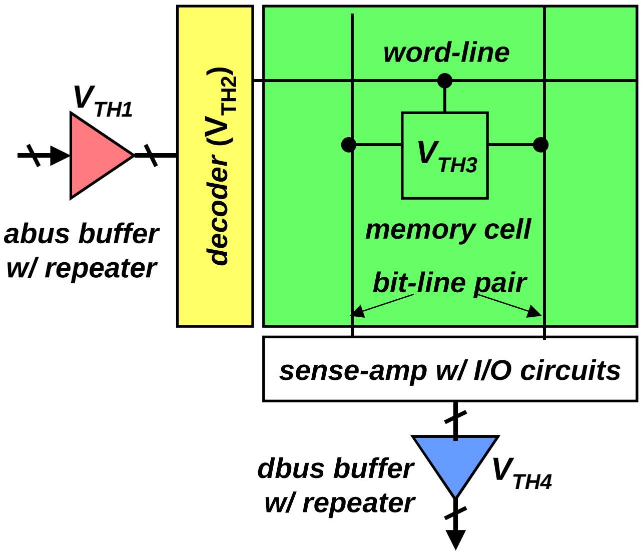


Figure 6.1: Cache sub-bank components.

Extensive HSPICE simulations were run to obtain leakage power and access time (or delay) models of wide ranges of cache sizes and V_{TH}'s for their four components. We considered V_{TH}'s between 200*mV* and 500*mV* in steps of 50*mV* at 1*V* nominal supply voltage. We measured the leakage power and access time of each cache component separately.

6.2.1 Leakage Power Models

Figure 6.2 shows V_{TH} vs. leakage power of the 7×128, 8×256, and 9×512 row decoders that we designed. The HSPICE simulation results shown in Figure 6.2 agree with the exponential decay in leakage power with the linear increase of V_{TH} that is characteristic of general CMOS circuits:

$$P_{leakage} \propto e^{-V_{TH}} \qquad \textit{(Eq. 6-1)}$$

To obtain an approximated analytic equation for leakage power as a function of V_{TH}, we measured the leakage power of the decoders at each discrete V_{TH} point, and we applied an

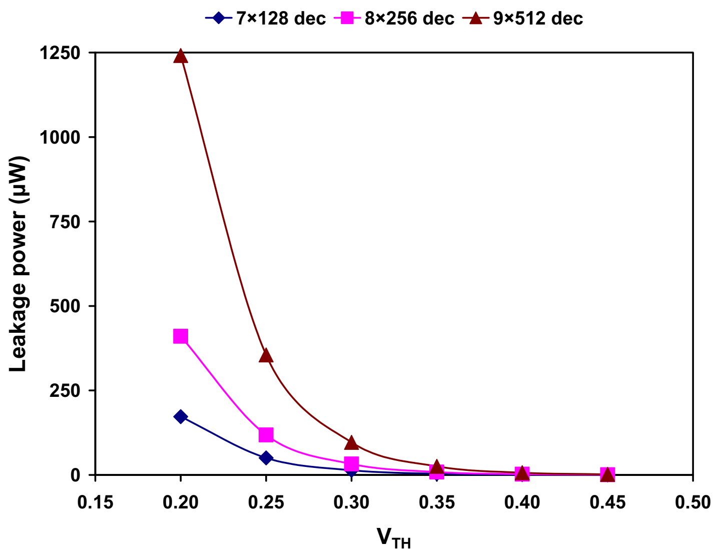


Figure 6.2: Leakage power dissipation of 7×128, 8×256, and 9×512 decoders.

exponentially decaying curve fitting technique to the measured leakage power as following:

$$P_{leakage}(V_{TH}) = A_0 + A_1 e^{-V_{TH}/a_1} \qquad \textit{(Eq. 6-2)}$$

where A_0, A_1, and a_1 are constants derived by using Origin 6.1, a scientific graphing and analysis software curve-fitting package [67]. Less than 0.001 *R-square error* is guaranteed for each fitted curves.

The rest of the cache components — address driver, data driver, and 6T SRAM cell array — show the same leakage power trend characteristics as the decoder in Figure 6.2; the leakage power decreases exponentially with the linear increase V_{TH}. Therefore, the identical curve-fitting technique can be applied for those components to derive approximated analytic leakage power models like Eq. (6-2). All the curve fitting coefficients are shown in Appendix C.

Once all the approximated analytic leakage power models for each component are derived for a cache size, the total leakage power of the cache can be approximated as a sum of the leakage power of all the components. Assuming that we can apply four distinct V_{TH}'s, the analytic approximated equation for leakage power, *LP* is:

$$LP(V_{TH1}, \ldots, V_{TH4}) = A_0 + A_1 e^{-V_{TH1}/a_1} + \ldots + A_4 e^{-V_{TH4}/a_4} \qquad \textit{(Eq. 6-3)}$$

where V_{TH1}, V_{TH2}, V_{TH3} and V_{TH4} represent the V_{TH}'s for address bus drivers, data bus drivers, decoders, and 6T-SRAM cell arrays, respectively. Each exponential term evaluates the leakage power dissipation of one of the four cache components.

6.2.2 Access Time Models

Figure 6.3 shows V_{TH} vs. delay time of the 7×128, 8×256, and 9×512 row decoders that we designed. Basically, the CMOS circuit delay of deep sub-micron short-channel transistors is:

$$T_{delay} = \frac{k \cdot L \cdot V_{DD}}{(V_{DD} - V_{TH})^{\alpha}} \qquad \textit{(Eq. 6-4)}$$

where *k*, *L*, and α[1] are constants depending on the technology and transistor sizes. The measured delay time trends in Figure 6.3 agree with the Eq. (6-4). However, the circuit delay or access time also fits very well to an exponential growth function with a very small exponent over our range of interest; it was also convenient for some of our optimizations to approximate delay this way.

1. α was around 2 in sub-micron technology, but it has been decreased to about 1.3 in the current generation deep sub-micron technology.

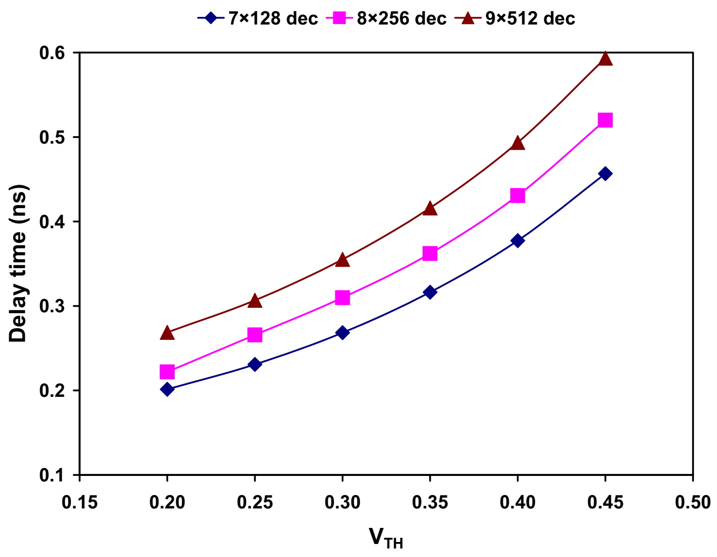


Figure 6.3: Delay time of 7×128, 8×256, and 9×512 decoders.

To obtain an approximated analytic equation for delay time as a function of V_{TH}, we measured the delay time of the decoders at each discrete V_{TH} point, and we applied an exponentially growing curve fitting technique to the measured delay time as following:

$$T_{delay}(V_{TH}) = B_0 + B_1 e^{-V_{TH}/b_1} \qquad \text{(Eq. 6-5)}$$

where B_0, B_1, and b_1 are constants derived from the same technique and R-square error range used for the leakage power models.

Similar to the decoder case, the rest of the cache components show the same delay trend characteristics for the linear increase of V_{TH} as shown in Figure 6.3. Hence, the same curve-fitting technique can be applied for those components to derive approximated analytic delay time models like Eq. (6-5) as functions of V_{TH}. All the curve-fitting coefficients are presented in Appendix C.

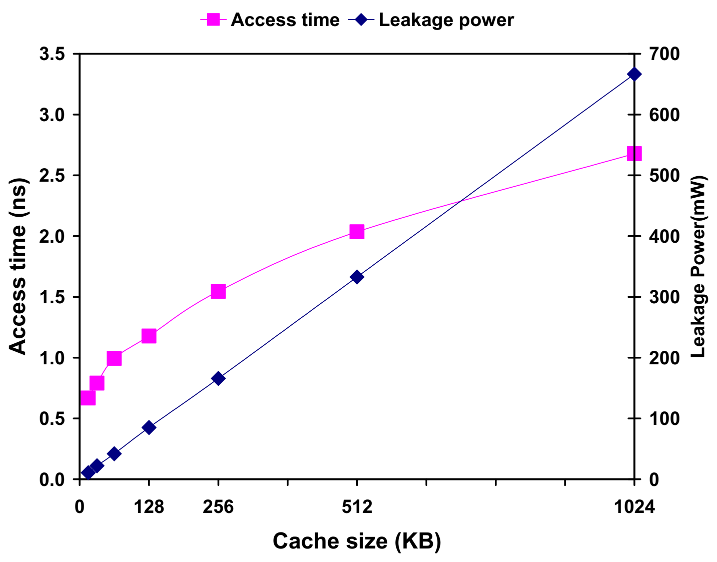
Once all the approximated analytic delay time models for each component are extracted for a cache size, total delay time or access time of the cache can be approximated as a sum of the delay time of all the cache components. Assuming that we can apply four distinct V_{TH}'s, the analytic approximated equation for the access time, *AT* is:

$$AT(V_{TH1}, \ldots, V_{TH4}) = B_0 + B_1 e^{V_{TH1}/b_1} + \ldots + B_4 e^{-V_{TH4}/b_4} \qquad \text{(Eq. 6-6)}$$

where V_{TH1}, V_{TH2}, V_{TH3} and V_{TH4} represent the V_{TH}'s for address bus drivers, data bus drivers, decoders, and 6T-SRAM cell arrays, respectively and each exponential term evaluates the delay time of of of the four cache component.

We also define baseline caches in which the V_{TH} of all the cache components is set to a low-V_{TH} (200*mV*). Figure 6.4 shows the access time and the leakage power of baseline caches. The cache access time grows logarithmically and the leakage power increases linearly with the cache size. Those trends agree with those of earlier studies on SRAM design. In Figure 6.4, we assume a direct-mapped cache organization and consider only the leakage power of data arrays.

Figure 6.4: Access time and leakage power vs. cache size of the baseline caches.



6.3 A Single Cache Leakage Optimization

6.3.1 Leakage Power Optimization with Multiple V_{TH} Assignments

In this section, we present a leakage power optimization technique assuming that we can assign multiple V_{TH}'s to a cache. To find the minimum leakage power of caches using maximum four distinct V_{TH}'s under a specified target access time constraint, AT, we formulate the problem as followings:

$$min\left\{LP(V_{TH1}, \ldots, V_{TH4}) = A_0 + A_1 e^{-V_{TH1}/a_1} + \ldots + A_4 e^{-V_{TH4}/a_4}\right\} \qquad \textit{(Eq. 6-7)}$$

$$\textit{constraints: } AT(V_{TH1}, \ldots, V_{TH4}) = B_0 + B_1 e^{V_{TH1}/b_1} + \ldots + B_4 e^{-V_{TH4}/b_4} \qquad \textit{(Eq. 6-8)}$$
$$0.2 \le V_{TH1}, V_{TH2}, V_{TH3}, V_{TH4} \le 0.5$$

where V_{TH1}, V_{TH2}, V_{TH3}, and V_{TH4} represent the V_{TH}'s for address bus drivers, data bus drivers, decoders, and 6T-SRAM arrays.

There exist numerous combinations of V_{TH1}, V_{TH2}, V_{TH3}, and V_{TH4} satisfying a specific target access time. Among those V_{TH} combinations, we find a quadruple of V_{TH1}, V_{TH2}, V_{TH3}, and V_{TH4} producing minimum leakage power using a numerical optimization method (e.g., Matlab's *fmincon* function); we considered any V_{TH} combination that satisfies a specified access time error range within 5%. We can then refine this to obtain an optimal set of V_{TH}'s with only 2 or 3 distinct V_{TH}'s by modifying the objective and constraint functions accordingly.

Assuming that we can assign distinct V_{TH}'s to each component of the cache, it is important to determine how many V_{TH}'s are cost-effective because an extra mask and process step are needed for each additional V_{TH}. To examine the dependence of the optimization results on access time, we sweep the target access time from the fastest possible (assigning a low V_{TH} (200*mV*) to all the cache components) to the slowest possible (assigning a high V_{TH} (500*mV*) to all the cache components). The following are the V_{TH} assignment schemes we examined in this study:

- *Scheme I*: assigning a high-V_{TH} to all the cache components including address bus drivers, data bus drivers, decoders, and 6T-SRAM cell arrays.
- *Scheme II*: assigning a high-V_{TH} only to 6T-SRAM cell arrays and assigning a default- or low-VTH (200*mV*) to the rest of the transistors.
- *Scheme III*: assigning a high-V_{TH} to 6T-SRAM cell arrays and assigning a different high-V_{TH} to the peripheral components — address bus drivers, data bus drivers, and decoders of the cache.
- *Scheme IV*: assigning four distinct high V_{TH}'s to all four cache components.

In Figure 6.5, we plot the normalized minimum leakage power and V_{TH} at different target access times (125%, 150%, 175%, and so forth) for 16KB caches of schemes I and II; the parenthesized *I* and *II* in Figure 6.3 represent the scheme I and II, respectively. In the graph, the normalized minimum leakage power and the access time of 100% correspond to the access time and the leakage power of a 16KB baseline cache designed with a low V_{TH} (200*mV*) for all the four cache components — the *fasted* but *leakiest* cache. The 125% access time in the *x*-axis means that the cache is 25% slower than the baseline cache.

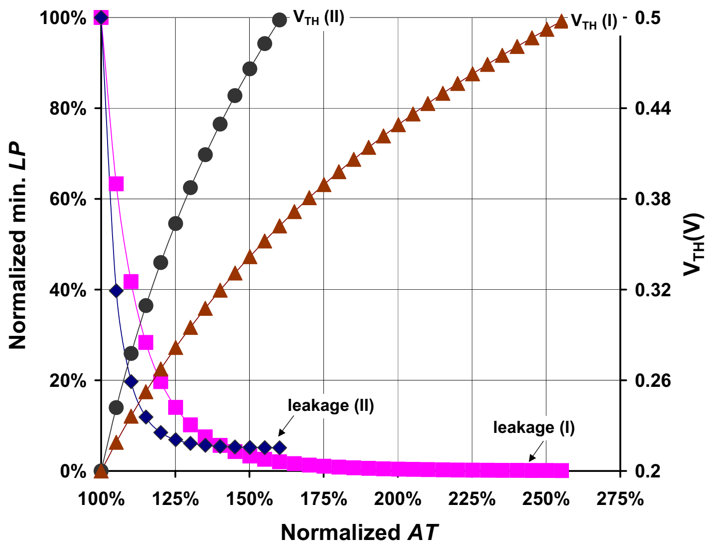


Figure 6.5: Normalized min. *LP* and V_{TH} vs. normalized *AT* — schemes I and II.

According to the trends shown in Figure 6.5, the leakage power decreases exponentially as the V_{TH} increases. The optimization results for the caches of different sizes show almost identical normalized minimum leakage power and V_{TH} trends to those of the 16KB caches in Figure 6.5 as long as the same V_{TH} assignment scheme is applied. Comparing two schemes, we can reduce more leakage power at the same access time point with the scheme II in the 100% ~ 140% normalized access time range. However, the scheme I shows better leakage power reduction beyond the 140% normalized access time point.

Figure 6.6 shows the normalized minimum leakage power and V_{TH} vs. normalized access time trends for a 16KB cache of scheme III. In Figure 6.6, the V_{TH} of SRAM cell arrays represented as *array* in the graph starts to increase first. This implies that the SRAM cell arrays that are responsible for the most significant fraction of total cache leakage power while it has the least impact on increasing the total cache access time. After the V_{TH} of the SRAM cell arrays are saturated to the maximum allowed point (500*mV*), the

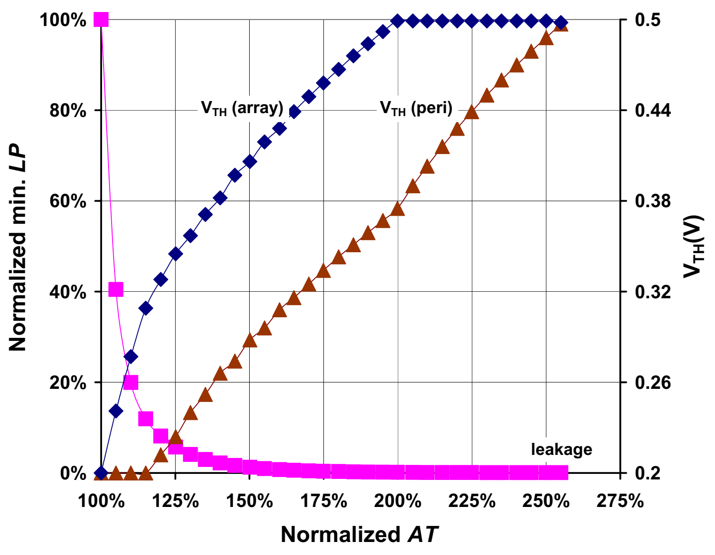


Figure 6.6: Normalized min. *LP* and V_{TH} vs. normalized *AT* — scheme III.

V_{TH} of the peripheral components labeled as *peri* in the graph is increased further to reduce more leakage power in the peripheral components. However, this just increases the access time without much further cache leakage reduction.

This leakage power and V_{TH} vs. access time trends also explain the leakage optimization results shown in Figure 6.6: scheme II shows a better optimization result in 100% ~ 140% normalized access time range than scheme I, but it does not beyond the 140% access time point. As stated, scheme I assigns a high-V_{TH} to all the cache components. It sacrifices more access time by increasing the V_{TH} of the peripheral components with less leakage reduction gain at the same access time point. But scheme II assigns the high-V_{TH} only to the SRAM cell arrays that are responsible for more significant fraction of total cache leakage power with less access time increase. However, scheme II cannot reduce leakage power further beyond the 140% access time point, because the leakage power by the peripheral components becomes substantial beyond this point.

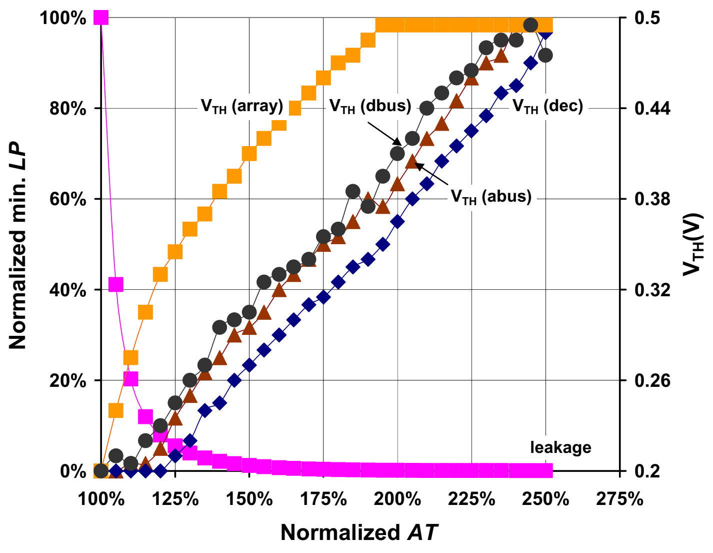


Figure 6.7: Normalized min. *LP* and V_{TH} vs. normalized *AT* — scheme IV.

Figure 6.7 shows the normalized minimum leakage power and V_{TH} vs. normalized access time trends for a 16KB cache of scheme IV. In scheme IV, we can assign up to 4 distinct V_{TH}'s for leakage power optimization. According to the results shown in Figure 6.7, the V_{TH} of 6T-SRAM cell arrays start to increase first similar to the scheme III case. Among the peripheral components, the V_{TH} for the decoder starts to increase last. This implies that the decoder has the least significant impact on the leakage power but it has the most significant impact on the overall cache access time; the address and data bus drivers show middling impact on both leakage power and access time compared to the 6T-SRAM cell array and the decoder. These V_{TH} trends suggest how cache components should be implemented to optimize direction for cache leakage power reduction.

Figure 6.8 compares the leakage power trends of schemes I, II, III, and IV for a 16KB cache. As expected, we can reduce more leakage power while achieving the same access time by having more V_{TH}'s to control. However, as the target access time is increased to more than the 140% point in scheme II, the cache dissipates more leakage

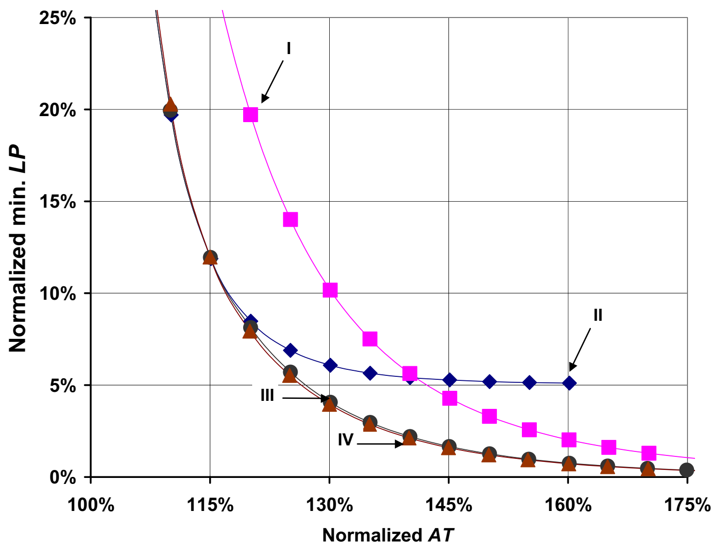


Figure 6.8: Normalized min. *LP* and V_{TH} vs. normalized *AT* — scheme IV.

power than one employing scheme I. This implies that the cache peripheral components consume non-negligible leakage power. Also, the leakage power by those components becomes substantial when we cut down the leakage power of the 6T-SRAM array significantly. Furthermore, the slowest cache access time point of scheme II ends around 150% for small size caches. This means that the peripheral components also play important roles in both cache leakage power and access time. In other words, increasing the V_{TH} of 6T-SRAM cell arrays alone gives a diminishing return at some point without reducing the leakage power further. This is why caches of scheme I give even better results than those of scheme II as V_{TH} increases.

Table 6.2 presents the normalized cache leakage power of schemes II, III, and IV to that of scheme I. The shaded numbers highlight the case such that caches of scheme II consume more leakage power than those of scheme I. The caches of scheme III and IV always show 38% ~ 74% better leakage optimization results than those of scheme I at the same constrained access time point. Other noticeable results are that there is a negligible

cache size (KB)	125% AT			150% AT			175% AT		
	II	III	IV	II	III	IV	II	III	IV
16	0.50	0.41	0.40	1.64	0.40	0.38	N/A	0.42	0.40
32	0.61	0.41	0.37	2.23	0.37	0.32	N/A	0.35	0.34
64	0.48	0.46	0.45	0.94	0.47	0.44	3.07	0.49	0.46
128	0.46	0.42	0.39	1.23	0.41	0.37	N/A	0.41	0.38
256	0.57	0.57	0.57	0.79	0.56	0.54	2.21	0.60	0.62
512	0.47	0.46	0.44	0.81	0.44	0.43	2.69	0.46	0.48
1024	0.33	0.32	0.30	0.82	0.28	0.26	N/A	0.30	0.29

Table 6.2: Normalized leakage power of scheme II, III, and IV to scheme I.

difference between caches of schemes III and IV in terms of leakage power reduction, which implies that 2 distinct high V_{TH}'s or caches of scheme III are enough for the leakage optimization.

6.3.2 Trade-Offs between Leakage Power and Access Times

One observation made from these experimental results is that we may not need to use very high V_{TH} (e.g., 500mV at 1V supply voltage) because this impedes the circuit speed unnecessarily without further leakage power reduction. Seeing the leakage reduction trends in Figure 6.5~Figure 6.8, we can reduce a significant amount of leakage power with only a modest increase of the access time (e.g., roughly up to ~120% of the normalized access time point).

Figure 6.9 shows a general trend in leakage power vs. access time. In the "fast / leaky" region, we can reduce the leakage power dramatically with a small sacrifice of the access time. On the other hand, we can hardly reduce the leakage power further by

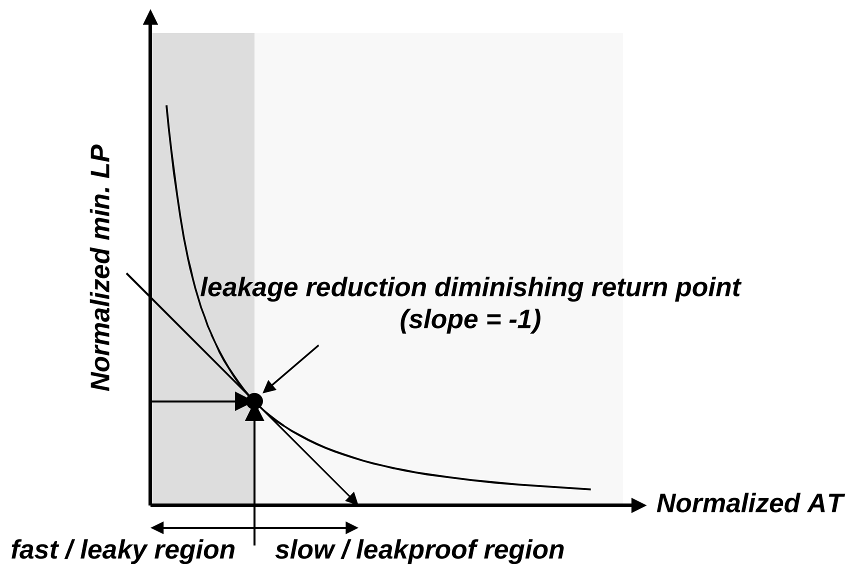


Figure 6.9: Leakage / access time trade-off point.

increasing V_{TH} after some point (e.g., 120% normalized access time point) while access time increases significantly.

Depending on the cache design goals or constraints such as size, access time, and static/dynamic power budgets, any point in Figure 6.9 can be chosen for the leakage power optimization. However, we can calculate a point whose tangential slope equals to negative “1” in the graph shown in Figure 6.9, and we call this the inflexion point of the cache leakage power and access time — in other words, this point can be regarded as an optimal leakage / access time trade-off point. This tangential slope should be obtained based on the normalized leakage power and access time; the normalized values have no associated unit.

Table 6.3 summarizes the normalized cache leakage power and the normalized access time at their inflexion point. This is also good indication for how many V_{TH}’s are good enough to control the leakage power effectively. The normalized access time and normalized leakage power are based on the leakiest leakage power and the fastest access

cache size (KB)	normalized AT				normalized LP			
	I	II	III	IV	I	II	III	IV
16	1.24	1.15	1.15	1.16	0.15	0.12	0.12	0.11
32	1.24	1.13	1.14	1.15	0.15	0.14	0.12	0.11
64	1.25	1.18	1.18	1.19	0.15	0.12	0.12	0.11
128	1.24	1.16	1.16	1.17	0.15	0.12	0.12	0.11
256	1.24	1.20	1.20	1.20	0.15	0.12	0.12	0.11
512	1.23	1.18	1.18	1.18	0.15	0.11	0.12	0.13
1024	1.23	1.15	1.15	1.16	0.14	0.09	0.09	0.09

Table 6.3: Normalized access time and leakage power at inflexion points.

time of caches designed with a low V_{TH} (200*mV*) for all the cache components. According to these results, we can reduce more leakage power at a faster access time point as we increase the number of distinct V_{TH}'s. Two high-VTH appears to be the "sweet spot." More than two does not buy much.

6.4 Two-Level Cache Leakage Optimization

In a microprocessor memory system, the average memory access time [69] is a key metric for measuring the overall memory system performance. To evaluate the performance, it is essential to examine the cache miss characteristics of realistic application programs because the performance is also strongly dependent on L1 and L2 cache miss rates as well as access times. In our study, we assume that the memory system hierarchy consists of separate L1 instruction and data caches with a unified L2 cache.

As stated in the beginning of this section, the overall memory system performance can be measured or compared with the *average memory access time* (AMAT) represented by:

$$\begin{aligned} AMAT &= Hit\ Time_{L1} + Miss\ Rate_{L1} \times \\ &(Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty\ Time_{L2}) \end{aligned} \qquad \textit{(Eq. 6-9)}$$

where *Miss Penalty Time*$_{L2}$ is the external memory access and data transfer time. In addition, the local miss rate[1] is used as the *Miss Rate*$_{L2}$. This local miss rate can be quite large for L2 caches because the total number of L2 cache accesses is equal to the total number of L1 cache misses, which is small.

Similarly, we define the *average memory access energy* (AMAE) to compare the dynamic energy dissipation of each memory system configuration. Assuming that the L1 cache is accessed every cycle, the AMAE represents the average energy dissipation per access in the entire microprocessor memory system that includes L1, L2 and main memory. We can estimate AMAE, as follow:

$$\begin{aligned} AMAE &= Hit\ Energy_{L1} + Miss\ Rate_{L1} \times \\ &(Hit\ Energy_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty\ Energy_{L2}) \end{aligned} \qquad \textit{(Eq. 6-10)}$$

where *Hit Energy* is average energy dissipation per access. For *Miss Penalty Energy*$_{L2}$, we assume a 2-channel 1066*Hz* 256MB RAMBUS DRAM RIMM module whose sustained transfer rate is 4.2*GB/s* [72] to derive the main memory access time and dynamic energy dissipation per access. Though the sustained transfer rate is quite high, we should also consider the RAS/CAS latency of the memory, which is about 20*ns*. For the energy dissipation per access, we used the number given in [73] — 3.57*nJ* per access.

To obtain L1 and L2 cache miss rates, we use the SimpleScalar / Alpha 3.0 tool set [68], a suite of functional and timing simulation tools for the Alpha AXP ISA. In addition, we collected the results from all 25 of the SPEC2K benchmarks [70] to perform our eval-

1. This rate is simply the number of misses in a cache divided by the total number of memory accesses to this cache.

L1 size (KB)	**miss rate (%)**	**L2 size (KB)**	**miss rate (%)**
16	3.3%	128KB	34.2%
		256KB	32.2%
		512KB	30.6%
		1024KB	25.5%
32	2.5%	256KB	40.3%
		512KB	38.2%
		1024KB	31.8%
64	1.5%	512KB	41.6%
		1024KB	35.6%

Table 6.4: L1 and L2 cache miss rates.

uation. (See Appendix B for the detailed microarchitectural simulation methodology.) We completed the execution for each benchmark application to get reliable L2 cache miss rates, because L2 cache accesses are far less frequent than L1 cache accesses; an insufficient number of L2 accesses may result in unrepresentatively higher L2 cache miss rates.

Table 6.4 shows the L1 and L2 cache miss rates for 16KB, 32KB, and 64KB L1 caches, respectively. We used direct-mapped L1 instruction caches and 4-way set associative L1 data caches. Also, we used 8-way set associative L2 caches. For simplicity, each L1 cache miss rate is obtained by the sum of the number of total instruction and data cache misses divided by the sum of total instruction and data cache accesses; a 16KB L1 means instruction and data caches are each 16KB in size. Since an L2 miss rate is a function of the L1 cache miss rate, we measure the separate L2 cache miss rates for each L1 cache size configuration. Those cache miss characteristics will definitely affect the leakage optimization direction of two-level cache memory system.

6.4.1 L2 Cache Leakage Power Optimization

Since a cache's contribution to leakage power is dominated by its size, we will examine the leakage power optimization of the L2 caches, first. Consider a conventional cache memory hierarchy of 16KB and 128KB for L1 and L2 caches respectively, designed with low-V_{TH} (0.2V) devices. With a fixed L1, reducing the leakage of the L2 by increasing V_{TH} makes the AMAT of the cache system becomes slower, because of the access time (or hit time) increase of the L2 cache. However, we are able to maintain the same AMAT and reduce the leakage power of the L2 by increasing its size to reduce its miss rate. The main memory access penalty is quite significant. Hence, even a slight reduction of L2 cache miss rates results in a significant improvement of the AMAT. We note that although area was one of the most important design constraints in the past, this trend is changing and power is becoming an equally important constraint in many situations [74].

Figure 6.10 shows the leakage power vs. AMAT of L2 caches with a fixed size L1 cache of 16KB. The leakage power optimization for individual caches is based on the scheme III requiring 2 distinct high V_{TH}'s. Assuming that the AMAT of a 128KB L2 cache as a base, we compare the leakage power of other caches at the same AMAT point; see the dotted vertical line in Figure 6.10. As can be seen from the plots, the AMAT can be maintained while the leakage power can be reduced by replacing a 128KB L2 with a 256KB L2 cache that is intentionally slowed down by increasing its V_{TH} to reduce leakage of the larger L2 cache.

This replacement with the double-sized L2 cache reduces the leakage power by 70%. If we employ a 512KB L2 cache, we can reduce the leakage power by 85% com-

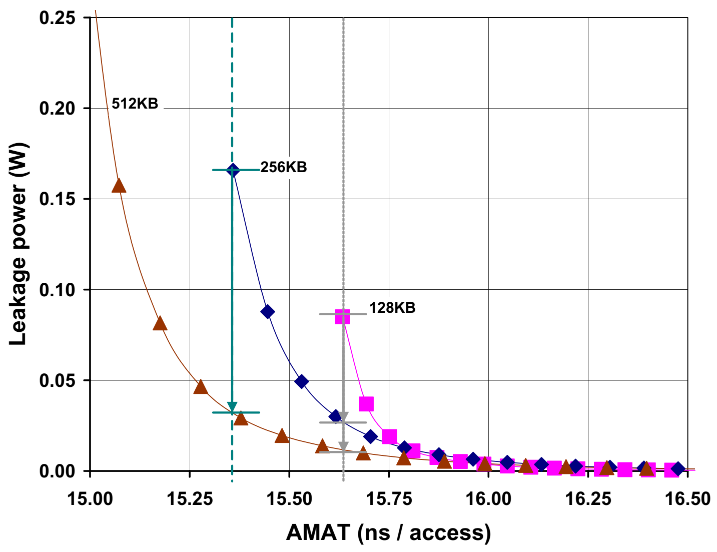


Figure 6.10: L2 leakage power optimization at the fixed L1 size.

pared to the fastest but leakiest 128KB L2 cache with the same AMAT. Similarly, the use of a 512KB L2 cache can further reduce leakage compared to the 256KB cache; see the dashed vertical line in Figure 6.10. As an alternative design space exploration, we can pick a point showing a faster AMAT and less leakage reduction with a larger L2 caches.

In addition, the employment of larger L2 caches also reduces the average dynamic power of the memory system because the larger L2 caches reduce the number of external memory accesses consuming a significant amount of dynamic energy. Table 6 summarizes the results for the normalized leakage power and normalized average memory access energy for each L1 cache size designed using scheme III at a fixed AMAT. To compare leakage power and AMAE, the following standard cache configurations were used: 128KB L2 with 16KB L1, 256KB L2 with 32KB L1, and 512KB L2 with 64KB L1. The shaded numbers represent the baseline L2 configuration, leakage power, and AMAE. Table 6 gives the somewhat counter intuitive results that we can reduce both leakage power and AMAE by employing larger L2 caches while maintaining a constant AMAT.

L1 size (KB)	L2 size (KB)	normalized leakage	normalized AMAE
16	128	1.00	1.00
	256	0.31	1.01
	512	0.15	0.99
32	256	1.00	1.00
	512	0.11	0.98
	1024	~0.00	0.89
64	512	1.00	1.00
	1024	0.01	0.95

Table 6.5: L2 cache normalized leakage and AMAE at the fixed L1 size.

6.4.2 L1 Cache Leakage Power Optimization

It is hard to improve the L1 cache miss rates further because they are already quite low for 16KB, 32K, and 64KB caches in the case of SPEC2K benchmarks. Hence, the access time of caches is the dominant factor in determining the AMAT. However, the access time of 64KB L1 cache can increase by 48% compared to a 16KB L1 cache, because access time is very sensitive to size for small caches. Essentially, cache access time increases logarithmically with size, but has a steeper slope for smaller caches than for larger caches. This observation confirms why the AMAT of a cache hierarchy with a smaller L1 cache can be faster than one with a larger L1 caches for a certain range of cache sizes (e.g., 16KB~64KB).

Figure 6.11 shows the leakage power vs. the AMAT of 16KB, 32KB, and 64KB L1 caches using scheme III each with a fixed L2 cache of size 512KB. Like the comparison performed in Section 6.4.1, the leakage power of different caches is compared at the

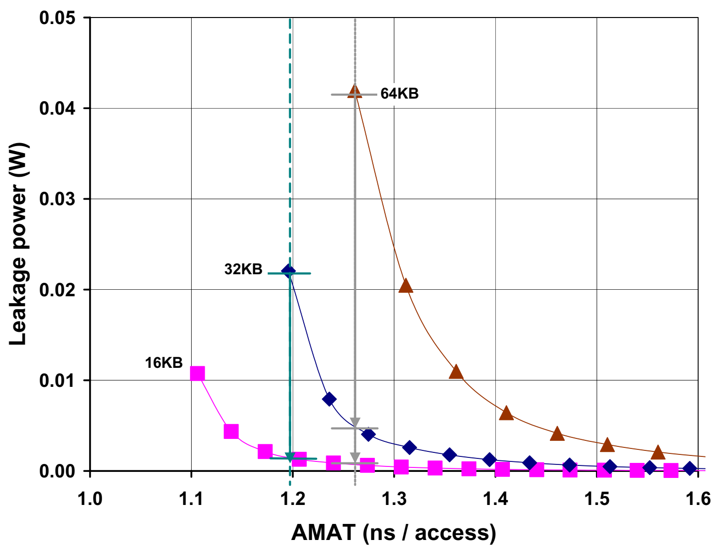


Figure 6.11: L1 leakage power optimization at the fixed L2 size.

same AMAT point. The plots show that leakage power can be reduced by replacing a 64KB L1 cache with a 32KB L1 cache that is intentionally slowed down by increasing its V_{TH}'s to reduce the leakage power, and the resulting cache memory system still has the same AMAT; see the dotted vertical line in Figure 6.11. Similarly, a slowed 16KB cache with increased V_{TH}'s can replace a 32KB without changing the AMAT of the L1/L2 hierarchy. The new system consumes much less leakage power; see the dashed vertical line in Figure 6.11.

Table 6.6 shows the results for normalized leakage power and AMAE for each fast but leaky L1 cache sizes using scheme III with fixed AMAT's. The comparisons were performed in the same manner as Table 6.6. The shaded numbers represent the baseline L2 configuration, leakage power, and AMAE. According to the comparisons, we can reduce both leakage power and AMAE by employing smaller L1 caches. This is the complement of the case for L2 caches, where the leakage of the overall memory system can be reduced by increasing their size.

L2 size (KB)	L1 size (KB)	normalized leakage	normalized AMAE
256	32	1.00	1.00
	16	0.06	0.95
512	64	1.00	1.00
	32	0.19	0.62
	16	0.02	0.59
1024	64	1.00	1.00
	32	0.12	0.64
	16	0.02	0.62

Table 6.6: L1 cache normalized leakage and AMAE at the fixed L2 size.

It should be noted, these results are only valid within the specific set of sizes and simulation environment given in this chapter. First, a 4KB L1 cache will have a cache miss rate that is much higher than a 16KB cache, but its access time will not be sufficiently smaller to make the trade-off worthwhile. Also, the normalized AMAE is rather high because the total power fraction of L1 caches is relatively small compared to L2 caches. Second, the cache foot-prints of the SPEC2K benchmark programs are fairly small compared to the real-world larger size applications. This results in quite high cache miss rates for small size L1 caches as shown in Table 6.4. Third, the operating system (OS) context-switching is not considered due to the limited simulation environment. The context switching is usually known to increase the cache miss rates because cache flushing[1] increases cold start misses. Hence, all those stated factors must be considered to obtain more reliable cache leakage power optimization results with the proposed techniques.

1. There are numerous cache management techniques for the efficient context switching other than this.

6.5 Chapter Summary

In this chapter, we examined the leakage power and access time trade-off trends where multiple V_{TH}'s are allowed. We used curve fitting techniques to model leakage power and access time. Our results show that 2 distinct V_{TH}'s for caches are sufficient to yield a significant reduction in leakage power. Such an arrangement can reduce the leakage power up to 91%; see the scheme III in Table 6.3 for an SRAM cache without significantly increasing access time. We also show that smaller L1 and larger L2 caches than are typical in today's processors result in significant leakage and dynamic power reduction without affecting the average memory access time. Given that the processor core may need a distinct V_{TH}, and each of the caches may need up to two V_{TH}'s (scheme III) we could require up to five distinct V_{TH}'s.

In this work, we assume that we have two-level on-chip caches. Recently, however, microprocessors with three-level caches are being deployed, and their L2 and L3 cache sizes are much larger than the caches discussed here. For future work, we will investigate leakage power optimization in a multi-level cache hierarchy that include L2 and L3 caches.

Chapter 7
Conclusion

As feature sizes shrink to increase integration density, threshold voltages decrease to reduce circuit delay, and more bits are integrated into on-chip caches to improve memory system performance, the leakage power of on-chip caches will dissipate a substantial amount of total microprocessor power consumption. In this thesis, various circuit and microarchitectural approaches were proposed to reduce the on-chip cache leakage power. They were evaluated and compared to previously proposed techniques.

7.1 Thesis Summary

First, we proposed a new low-leakage, state preserving static random access memory architecture. This technique reduces the leakage power by 94% at a 0.25V stand-by voltage with 1 or 2 cycle wake-up latency. It also requires only the addition of a small amount of voltage control circuitry. The estimated extra area required to implement the voltage control circuit used to wake up a 128-bit cache line in 1 cycle is approximately 1.48%.

Second, we proposed the drowsy L1 data caches with the simple policy of periodically putting the entire cache lines into drowsy mode and we demonstrated that this simple policy does about as well as a policy that tracks access history of cache lines. The proposed cache system reduces the L1 data cache leakage power by 80% with less than 1% run-time increase.

Third, we found out that the policy used for data caches does not work well for instruction caches. After the analyses of instruction cache working sets, we concluded that the noaccess policy is more suitable for instruction caches. We also proposed an early wake-up technique. This reduces the average run-time increase of the noaccess policy without the early wake-up by more than 40%. The proposed technique reduces the L1 instruction cache leakage power by 60% with negligible (0.07%) run-time increase.

To further reduce the leakage power from the supply voltage source to the bit-lines, we also proposed a predictive gated bit-line precharge technique. When combined with the early wake-up noaccess policy, it reduces leakage power by a further 13% with an average 0.3% run-time increase. Compared to the "cache decay" technique, the proposed instruction caches and data caches show less run-time increase and more leakage savings overall. The weakness of the cache decay technique is that it may increase the overall energy consumption, because it incurs additional upper-level memory system accesses, which results in significant dynamic power dissipation. In addition, this technique is sensitive to the amount of energy consumption by the upper-level cache system and the update window size — an incorrect decision about turning off caches incurs the additional upper-level cache accesses.

Finally, we proposed the leakage power optimization techniques for multi-level caches at the circuit and microarchitectural boundary. To investigate the optimization techniques, we developed leakage power and access time models as functions of the threshold voltages and cache sizes. With those models, we performed various leakage power and memory system performance trade-off studies.

In most cache sizes, the optimization results reduce the leakage power more than 80% with less than 20% access time increases. Furthermore, implementing larger on-chip L2 caches with smaller L1 caches can significantly reduce both the static leakage and dynamic power at the same time. For instance, using a leakage optimized 512KB L2 cache instead of an unoptimized 256KB one with a 32KB L1 caches for both cases, reduces the L2 cache leakage power by 90% while maintaining the same average memory system performance. In addition, using a leakage optimized 16KB L1 cache instead of an unoptimized 32KB one with a 256KB L2 cache for both cases, decreases the L1 cache leakage power by 94% while maintaining the same average memory system performance.

7.2 Future Directions

7.2.1 Low-Leakage and State-Preserving 6-Transistor Memory

The proposed low-leakage, state-preserving memory circuit increases the noise susceptibility from cross-coupling and particle strikes due to the reduced supply voltage used for the memory cells in drowsy mode. Although the probability of those errors is extremely small, the charge stored in a cell decreases as the feature size shrinks. This increases the soft error rate even for memory cells in the active mode. Therefore, to prevent a system crash from corrupted bits, we need to consider including error correction mechanisms in each cache line of L1 caches.

Finally, the proposed circuit technique requires an additional power supply source for the drowsy mode. The stand-by voltage is lower than the nominal supply voltage and it will thus be more affected by power supply line fluctuation and IR drops. Instead of using dual supply voltage sources, we can generate a stand-by voltage from the nominal supply voltage source with a more sophisticated form of voltage controller implemented in each cache line.

7.2.2 Microarchitectural Controls for Low-Leakage Caches

First, an open question remains as to the role of adaptability in determining the window size. We found that for a given machine configuration, a single static window size of 4K cycles performs adequately on all of the SPEC2K benchmarks. However, the optimum varies slightly for each workload, thus making the window size adaptive would allow a finer power-performance trade-off. Furthermore, in the same workload, we may need a different update window size depending on the workload execution phase.

Second, a major drawback of the drowsy technique for instruction caches was non-negligible performance losses for some workloads. But we proved that a simple next-line early wake-up was quite effective at reducing the performance losses. Hence, to further reduce the performance losses, we can use a more sophisticated next-line early wake-up technique based existing prefetching techniques. This will reduce the run-time increase, as well as allow us to use a smaller update window size to more leakage power even further.

Finally, we can refine the predictive gated precharge technique by using a more sophisticated predictor, although the current predictor performs very well for most workloads. One may consider a state-of-the-art branch predictor technique.

7.2.3 Leakage Optimization via Multiple Threshold Voltage Assignments

First, the proposed leakage optimization technique applies multiple threshold voltages at a component (e.g., decoder block) granularity; one component is allowed to have only one threshold voltage. However, to further optimize the circuit speed, we can selectively employ the high threshold devices within the components. To speed up the circuit, low threshold voltage transistors can be used for the circuits on the critical path of the component while high threshold voltage ones are used for those not on the critical path. This will require a more complex leakage power and circuit delay models, but will yield further reduction in overall leakage

Second, we can apply cycle-accurate microarchitectural simulators to leakage optimized multi-level caches to examine the memory system performance after the optimization. To do this, we need a more accurate model converting the access time of the designed cache to the equivalent number of L2 cache access latency cycles, because the number of cycles of the L2 cache access latency involves the bus-width between the L1 and L2 caches, and the memory protocol between the caches.

7.2.4 Gate Oxide Leakage

During the final phase of the research described in this thesis, the effect of gate oxide leakage becomes a concern that rivalled sub-threshold leakage. A recent publication of ours details this trend [80]. Many of the optimization techniques developed here can be "replayed" with gate oxide thickness as the design parameters. This remains an interesting goal for further study.

Appendix A
Supply Voltage Rail Wire Parameters

The estimated height and width of one memory cell in 70nm technology is 1.42µm by 0.72µm and 128 memory cells are connected to a supply voltage rail wire.

Table A.1: The supply voltage rail wire dimension parameters.

width	space	length	thickness	height
0.14µm	1.42µm	183.18µm	0.35µm	0.20µm

The wire interconnect capacitance and resistance in Table A.2 are derived from [4] with the given wire dimensions in Table A.1.

Table A.2: The supply voltage rail wire interconnect parameters.

capacitance ($K_{dielectric}$ = 2.2)			resistance	
C_{ground}	C_{couple}	C_{total}	R	R_{total}
21.30fP	2.30fP	25.89fP	408.16fP	74.77

Appendix B
Processor Simulation Methodology and Results

The architectural simulator used in this study is derived from the SimpleScalar/ Alpha 3.0 tool set [68], a suite of functional and timing simulation tools for the Alpha AXP ISA. Simulation is execution-driven, including execution down any speculative path until the detection of a fault, TLB miss, or branch misprediction. Specifically, we extended *sim-outorder* to reflect the performance impact of waking up the drowsy cache lines or sub-banks in the L1 data and instruction caches. The processor simulation parameters are listed in Table B.1. The processor microarchitectural parameters model a high-end microprocessor similar to an Alpha 21264. We augment it with a generous supply of functional units, aggressive main memory, L1 caches, and a register file with a latency to reduce the execution variability due to resource constraints and memory latencies. To perform our evaluation we collected results from all 25 of the SPEC2000 benchmarks [70].

Table B.1: Processor simulation parameters.

Out of Order Execution	4-wide fetch / decode / issue / commit, 64 RUU, 32 LSQ, speculative scheduling
Functional Unit (latencies)	4 integer ALUs (1), 2 floating point ALUs (2), 1 integer MULT/DIV (3/20), 1 floating point MULT/DIV/SQRT (4/12/24), 2 general memory ports
Branch Prediction	combined bimodal (4K-entry) / gshare (4K-entry) w/ selector (4K-entry), 32-entry RAS, 512-entry 4-way BTB, at least 11 cycles for branch misprediction recovery
Memory System (latencies)	32KB 2-way 32-byte block L1 inst (1) and data caches (1), 512KB 4-way 64-byte block unified L2 cache (12), 128-entry fully associative inst and data TLB (28/28) main memory (80/8)

All SPEC programs were compiled for a Compaq Alpha AXP-21264 processor using the Compaq C and Fortran compilers under the OSF/1 V4.0 operating system using full compiler optimizations (-O4). The simulations were run for 100 million instructions using the SPEC reference inputs. We used the utility Early *SimPoints* [71] to pinpoint program locations of peak performance so that we can find simulation region that most stress in particular instruction and data caches.

In addition, we present the full SPEC2K benchmark microarchitectural simulation results for the experiments in the main thesis chapters, because only a sub-set of the results were shown to reduce clutter in the figures and tables.

Table B.2: Data cache working set characteristics.

benchmark	working set	number of accesses	accesses / line	accesses / cycle
bzip2	7%	820.5	10.9	0.40
crafty	16%	1313.9	8.0	0.64
eon	15%	1685.7	10.7	0.82
gap	5%	1531.8	28.3	0.75
gcc	34%	2898.8	8.4	1.42
gzip	8%	1264.5	15.0	0.62
mcf	8%	264.4	3.3	0.13
parser	8%	894.4	10.3	0.44
perl	10%	842.4	8.3	0.41
twolf	8%	476.0	5.5	0.23
vortex	11%	1319.6	11.4	0.64
vpr	11%	361.5	12.6	0.66
ammp	8%	680.3	8.6	0.33
applu	12%	833.6	6.8	0.41
apsi	10%	1115.1	11.0	0.54
art	14%	336.0	2.4	0.16
equake	6%	1956.2	29.7	0.96
facerec	7%	884.6	13.2	0.43
fma3d	5%	787.6	14.1	0.38
galgel	34%	1594.6	4.6	0.78
lucas	8%	385.1	4.7	0.19
mesa	8%	1452.0	17.2	0.71
mgrid	16%	907.2	5.6	0.44
sixtrack	15%	980.5	6.2	0.48
swim	12%	368.2	2.9	0.18
wupwise	9%	1181.3	13.2	0.58
AVG	12%	1082.2	8.9	0.40

A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used.

Table B.3: Data cache working set reuse characteristics.

benchmark	n=1	n=2	n=8	n=32
bzip2	34%	19%	8%	3%
crafty	61%	47%	24%	9%
eon	64%	51%	40%	25%
gap	52%	28%	17%	11%
gcc	3%	0%	0%	0%
gzip	39%	30%	23%	17%
mcf	8%	4%	2%	2%
parser	35%	26%	17%	10%
perl	33%	21%	13%	10%
twolf	36%	15%	7%	3%
vortex	54%	36%	10%	5%
vpr	52%	39%	20%	11%
ammp	25%	15%	7%	1%
applu	19%	2%	2%	2%
apsi	50%	30%	25%	23%
art	8%	1%	1%	1%
equake	98%	97%	95%	95%
facerec	39%	26%	13%	9%
fma3d	38%	14%	2%	0%
galgel	52%	23%	0%	0%
lucas	19%	5%	5%	5%
mesa	77%	67%	61%	54%
mgrid	32%	10%	3%	2%
sixtrack	64%	56%	41%	25%
swim	41%	0%	0%	0%
wupwise	56%	47%	38%	29%
AVG	40%	25%	15%	11%

A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used. The fractions of accesses that are the same as in the n-th previous window in the current 2K cycle window size.

Table B.4: Instruction cache working set characteristics.

benchmark	working set	number of accesses	accesses / line	accesses / cycle
bzip2	3%	949.8	28.5	0.46
crafty	28%	1285.2	4.5	0.63
eon	26%	1386.5	5.3	0.68
gap	15%	1492.3	9.6	0.73
gcc	1%	1342.7	137.1	0.66
gzip	3%	1467.5	41.0	0.72
mcf	1%	225.1	34.4	0.11
parser	4%	993.4	25.2	0.49
perl	18%	742.4	4.0	0.36
twolf	9%	619.7	6.4	0.30
vortex	21%	994.7	4.6	0.49
vpr	4%	1117.8	25.3	0.55
ammp	3%	590.1	17.7	0.29
applu	7%	679.4	9.0	0.33
apsi	16%	1096.4	6.7	0.54
art	1%	423.2	44.1	0.21
equake	18%	1874.6	10.0	0.92
facerec	3%	941.1	35.6	0.46
fma3d	14%	1427.7	10.0	0.70
galgel	1%	1133.3	214.7	0.55
lucas	2%	655.5	30.2	0.32
mesa	17%	1367.7	7.7	0.67
mgrid	13%	694.2	5.3	0.34
sixtrack	13%	1030.7	8.0	0.50
swim	6%	335.8	5.7	0.16
wupwise	13%	1285.3	9.4	0.63
AVG	10%	1005.9	9.8	0.38

A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used.

Table B.5: Instruction cache working set reuse characteristics.

benchmark	n=1	n=2	n=8	n=32
bzip2	81%	70%	41%	15%
crafty	64%	49%	20%	3%
eon	58%	41%	31%	20%
gap	49%	31%	17%	10%
gcc	71%	62%	45%	0%
gzip	96%	94%	87%	76%
mcf	93%	90%	83%	75%
parser	92%	87%	73%	53%
perl	33%	19%	7%	4%
twolf	16%	3%	0%	0%
vortex	43%	25%	5%	1%
vpr	95%	92%	76%	50%
ammp	77%	63%	27%	1%
applu	99%	99%	99%	99%
apsi	73%	63%	54%	50%
art	99%	99%	97%	91%
equake	99%	98%	98%	98%
facerec	82%	72%	61%	59%
fma3d	21%	5%	0%	0%
galgel	91%	82%	59%	57%
lucas	100%	100%	100%	100%
mesa	92%	86%	76%	64%
mgrid	76%	73%	72%	72%
sixtrack	98%	96%	91%	83%
swim	99%	99%	99%	99%
wupwise	80%	74%	59%	54%
AVG	66%	57%	45%	38%

A 32KB, 2-way set associative cache with 1024 32-byte lines and a 2048-cycle update window are used. The fractions of accesses that are the same as in the n-th previous window in the current 2K cycle window size.

Table B.6: Run-time increase and leakage reduction comparison.

benchmark	run-time increase (%)		normalized leakage (%)	
	instruction	data	instruction	data
bzip2	1.06	0.77	96	93
crafty	11.49	0.46	65	86
eon	10.79	0.36	65	88
gap	6.32	0.46	79	95
gcc	0.10	0.06	99	66
gzip	1.15	0.89	96	93
mcf	0.05	0.66	99	93
parser	1.34	1.17	96	92
perl	8.47	0.87	73	91
twolf	2.45	0.87	83	92
vortex	8.58	0.25	70	90
vpr	1.10	0.89	95	90
ammp	0.56	0.54	96	93
applu	0.52	0.50	93	89
apsi	7.44	0.32	81	91
art	0.03	0.61	99	87
equake	8.96	0.03	82	94
facerec	1.59	0.15	97	95
fma3d	0.11	0.66	98	93
galgel	0.08	0.31	99	70
lucas	2.74	0.57	87	89
mesa	6.27	0.21	81	93
mgrid	3.18	0.49	84	85
sixtrack	2.74	0.34	87	87
swim	0.15	0.61	94	89
wupwise	7.15	0.20	84	92
AVG	2.74	0.57	87	89

We use a simple policy with 1-cycle drowsy-line wake-up latency and 4K-cycle update window size for both the 32KB 2-way set associative instruction and data caches.

Table B.7: Run-time increases of on-demand gated precharge.

benchmark	run-time increases		
	8×4KB sub-banks	4×8KB sub-banks	2×8KB sub-banks
bzip2	2.0%	0.8%	0.8%
crafty	16.6%	16.1%	14.4%
eon	7.3%	5.0%	3.2%
gap	7.4%	6.8%	5.1%
gcc	0.1%	0.0%	0.0%
gzip	6.1%	3.9%	3.9%
mcf	0.7%	0.7%	0.0%
parser	2.1%	1.6%	1.4%
perl	8.8%	8.2%	7.7%
twolf	1.2%	1.0%	0.6%
vortex	16.4%	15.0%	12.9%
vpr	0.6%	0.6%	0.0%
ammp	0.8%	0.8%	0.0%
applu	0.0%	0.0%	0.0%
apsi	8.4%	7.4%	7.0%
art	0.0%	0.0%	0.0%
equake	17.8%	14.3%	7.1%
facerec	1.9%	1.9%	1.3%
fma3d	23.3%	21.7%	19.1%
galgel	0.0%	0.0%	0.0%
lucas	0.0%	0.0%	0.0%
mesa	14.5%	13.8%	11.3%
mgrid	0.2%	0.2%	0.2%
sixtrack	4.7%	3.7%	2.7%
swim	0.0%	0.0%	0.0%
wupwise	1.2%	0.2%	0.2%
AVG	3.8%	3.3%	2.6%

We use the 32KB, 2-way set associative instruction and data caches.

Appendix C

Cache Leakage and Delay Model Coefficients

Table C.1: Address bus leakage and delay model coefficients.

cache size (KB)	leakage			delay		
	A_0	A_1 ($\times10^{-3}$)	a_1 ($\times10^{-3}$)	B_0 ($\times10^{-12}$)	B_1 ($\times10^{-12}$)	b_1 ($\times10^{-3}$)
16	0.00	34.33	40.86	74.00	29.72	218.76
32	0.00	71.61	40.90	106.92	39.29	208.18
64	0.00	39.24	40.86	89.02	32.82	216.49
128	0.00	81.84	40.90	145.75	42.90	197.87
256	0.00	44.14	40.86	133.63	35.89	207.89
512	0.00	92.07	40.90	254.46	43.79	179.25
1024	0.00	187.92	40.91	486.24	66.96	166.30

Table C.2: Decoder leakage and delay model coefficients.

cache size (KB)	leakage			delay		
	A_0	A_1 ($\times10^{-3}$)	a_1 ($\times10^{-3}$)	B_0 ($\times10^{-12}$)	B_1 ($\times10^{-12}$)	b_1 ($\times10^{-3}$)
16	0.00	97.39	40.41	94.524	37.246	199.49
32	0.00	194.82	40.41	94.524	37.246	199.49
64	0.00	243.70	40.00	67.932	54.359	218.48
128	0.00	487.46	40.00	67.932	54.359	218.48
256	0.00	765.56	39.70	130.71	52.610	207.02
512	0.00	1531.12	39.70	130.71	52.610	207.02
1024	0.00	3062.24	39.70	130.71	52.610	207.02

Table C.3: SRAM array leakage and delay model coefficients.

cache size (KB)	leakage			delay		
	A_0	A_1 $(\times 10^{-3})$	a_1 $(\times 10^{-3})$	B_0 $(\times 10^{-12})$	B_1 $(\times 10^{-12})$	b_1 $(\times 10^{-3})$
16	0.00	1.25	41.60	60.98	103.17	279.83
32	0.00	2.50	41.60	60.98	103.17	279.83
64	0.00	5.01	41.58	121.96	206.34	279.83
128	0.00	10.01	41.58	121.96	206.34	279.83
256	0.00	19.86	41.66	243.92	412.69	279.83
512	0.00	39.72	41.66	243.92	412.69	279.83
1024	0.00	79.44	41.66	243.92	412.69	279.83

Table C.4: Data bus leakage and delay model coefficients.

cache size (KB)	leakage			delay		
	A_0	A_1 $(\times 10^{-3})$	a_1 $(\times 10^{-3})$	B_0 $(\times 10^{-12})$	B_1 $(\times 10^{-12})$	b_1 $(\times 10^{-3})$
16	0.00	13.55	41.62	26.30	10.25	221.36
32	0.00	54.18	41.62	58.93	20.04	201.37
64	0.00	54.18	41.62	41.32	13.34	215.10
128	0.00	216.73	41.62	96.97	24.27	188.59
256	0.00	108.35	41.62	85.62	16.67	199.68
512	0.00	433.40	41.62	203.24	26.98	168.73
1024	0.00	1083.54	41.62	433.07	51.53	160.31

Reference

[1] R. Wilson and D. Lammers, “Grove Calls Leakage Chip Designers’ Top Problem,” EE Times, 13 Dec. 2002, http://www.eetimes.com/story/ OEG20021213S0040.

[2] Semiconductor Industry Association, “International Technology Roadmap for Semiconductors,” 2002, http://public.itrs.net.

[3] B. Doyle, et al., “Transistor Elements for 30nm Physical Gate Lengths and Beyond,” *Intel® Technology Journal*, Vol 6, Page(s): 42~54, May 2002.

[4] Berkeley Predictive Technology Model, http://www-device.eecs.berke-ley.edu.

[5] K. Ghose and M. Kamble, “Reducing Power in Superscalar Processor Cach-es using Subbanking, Multiple Line Buffers and Bit-line segmentation”, *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 70~75, Aug 1999.

[6] N. S. Kim, D. Blaauw, and T. Mudge, “Leakage Power Optimization Tech-niques for Ultra Deep Sub-Micron Multi-Level Caches,” *Proc. IEEE/ACM Int. Conf. on Computer Aided Desgn*, Page(s): 627~632, Nov 2003.

[7] S. Mutoh, et al., “1-V Power Supply High-Speed Digital Circuit Technolo-gy with Multithreshold-Voltage CMOS,” *IEEE Journal of Solid-State Cir-cuits*, Vol 30, Page(s): 847~854, Aug 1995.

[8] N. Shibata, H. Morimura, and M. Harada, “1-V 100-MHz Embedded SRAM Techniques for Battery-Operated MTCMOS/SIMOX ASICs,” *IEEE Journal of Solid-State Circuits*, Vol 3, Page(s): 1396~1407, Oct 2000.

[9] T. Douseki, N. Shibata, and J. Yamada, “A 0.5-1V MTCMOS/SIMOX SRAM Macro with Multi-Vth Memory Cells,” *IEEE Int. SOI Conf.*, Page(s): 24~25, Oct 2000.

[10] H. Mizuno, et al., “A 18μA-Standby-Current 1.8V 200MHz Microproces-sor with Self Substrate-Biased Data-Retention Mode,” *IEEE Int. Solid-State Circuit Conf.*, Page(s): 280~281, Feb 1999.

[11] H. Mizuno, et al., “An 18-μA Standby Current 1.8V, 200MHz Microproces-sor with Self-Substrate-Biased Data-Retention Mode,” *IEEE Journal of Solid-State Circuits*, Vol 34, Page(s): 1492~1500, Nov 1999.

[12] X. Liu, and S. Mourad, “Performance of Submicron CMOS Devices and Gates with Substrate Biasing,” *IEEE Int. Symp. on Circuits and Systems*, Vol 4, Page(s): 9~12, May 2000.

[13] A. Keshavarzi, et. al., “Effectiveness of Reverse Body Bias for Leakage Control in Scaled Dual Vt CMOS ICs,” *IEEE/ACM Int. Symp. on Low Pow-*

er Electronic and Device, Page(s): 207~212, Aug 2001.

[14] M. Miyazaki, J. Kao, and A. P. Chandrakasan, “A 175 mV Multiply-Accumulate Unit Using An Adaptive Supply Voltage and Body Bias (ASB) Architecture,” *IEEE Int. Solid-State Circuits Conf.*, Page(s): 58~59, Feb 2002.

[15] M. Miyazaki, J. Kao, and A. P. Chandrakasan, “A 175-mV Multiply-Accumulate Unit Using An Adaptive Supply Voltage and Body Bias Architecture,” *IEEE Journal of Solid-State Circuits*, Vol 37, Page(s) 1545~1554, Nov 2002.

[16] S. Narendra, et al., “1.1V 1GHz Communications Router with On-Chip Body Bias in 150nm CMOS,” *IEEE Int. Solid-State Circuits Conf.*, Page(s): 218~219, Feb 2002.

[17] J. Tschanz, et al., “Effectiveness of Adaptive Supply Voltage and Body Bias for Reducing Impact of Parameter Variations in Low Power and High Performance Microprocessors,” *IEEE Int. Symp. on VLSI Circuits*, Page(s): 310~311, Jun 2002.

[18] P. Ko, et al., “BSIM3 for Analog and Digital Circuit Simulation,” *IEEE Int. Symp. on VLSI Tech. CAD*, Page(s): 400~429, Jan 1993.

[19] Z. H. Liu, et al., “Threshold Voltage Model for Deep-Submicrometer MOSFETs,” *IEEE Transaction on Electron Devices*, Vol 40, Page(s): 86~95, 1993.

[20] K. Nii, et al., “A Low Power SRAM using Auto-Backgate-Controlled MT-CMOS,” *IEEE/ACM Int. Symp. on Low Power Electronic and Device*, Page(s): 293~298, Aug 1998.

[21] C. H. Kim, K. Roy, “Dynamic Vt SRAM: A Leakage Tolerant Cache Memory for Low Voltage Microprocessors,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 251~254, Aug 2002.

[22] C. H. Kim, et al., “A Forward Body-Biased Low-Leakage SRAM Cache: Device and Architecture Considerations,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 6~9, Aug 2003.

[23] M. Ketkar, and S. Sapatnekar, “Standby Power Optimization via Transistor Sizing and Dual Threshold Voltage Assignment,” *IEEE Int. Conf. on Computer Aided Design*, Page(s): 375~378, Nov 2002.

[24] J. Kao, and A. Chandrakasan, “Dual-Threshold Voltage Techniques for Low-Power Digital Circuits,” *IEEE Journal of Solid-State Circuits*, Vol 35, Page(s): 1009~1018, Jul 2002.

[25] A. Keshavarzi, and S. Ma, “Effectiveness of Reverse body Bias for Leakage Control in Scaled Dual Vt CMOS ICs,” *IEEE/ACM Int. Symp. Low Power Electronic and Device*, Page(s): 207~212, Aug 2001.

[26] T. Karnik, et al., “Total Power Optimization by Simultaneous Dual-Vt Allocation and Device Sizing in High Performance Microprocessors,” *IEEE/*

ACM 39th Design Automation Conf., Page(s): 486~491, Jun 2002.

[27] I. Fukushi, et al., “A Low-Power SRAM using Improved Charge Transfer Sense Amplifiers and A Dual-Vth CMOS Circuit Scheme,” *IEEE Int. Symp. on VLSI Circuits,* Page(s): 142~145, Jun 1998.

[28] K. Itoh, et al., “A Deep Sub-V, Single Power-Supply SRAM Cell with Multi-VT, Boosted Storage Node and Dynamic load,” *IEEE Int. Symp. on VLSI Circuits*, Page(s): 132~133, Jun 1996.

[29] F. Hamzaoglu, et al., “Dual-VT SRAM Cells with Full-Swing Single-Ended Bit Line Sensing for High-Performance On-Chip Cache in 0.13 μm technology generation,” *IEEE/ACM Int. Symp. Low Power Electronics and Device,* Page(s): 15~19, Aug 2000.

[30] F. Hamzaoglu, et al., “Analysis of Dual-VT SRAM Cells with Full-Swing Single-Ended Bit Line Sensing for On-Chip Cache,” *IEEE Transaction on Very Large Scale Integration (VLSI) Systems,* Vol 10, Page(s): 91~95, Apr 2002.

[31] M. Powell, et al., “Gated-V_{DD}: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design,* Page(s): 90~95, Aug 2000.

[32] Y. Ye, S. Borkar and V. De, “A Technique for standby leakage reduction in high-performance circuits,” *IEEE Int. Symp. on VLSI Circuits*, Page(s): 40~41, Jun 1998.

[33] J. Halter and F. Najm, “A Gate-Level Leakage Power Reduction Method for Ultra-Low-Power CMOS Circuits,” *IEEE Custom Integrated Circuits Conf.*, Page(s): 475~478, Sep 1997.

[34] Z. Chen, M. Johnson, et al., “Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 239~244, Aug 1998.

[35] S. Narendra, et al., “Scaling of stack effect and its application for leakage Reduction,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 195~200, Aug 2001.

[36] A. Agarwal, et al., “A Data Retention Gated-Ground Cache for Low Power”, *IEEE/ACM 39th Design Automation Conf.*, Page(s): 473~478, Jun 2002.

[37] A. Agarwal, L. Hai, and K. Roy, “A Single-Vt Low-Leakage Gated-Ground Cache for Deep Submicron,” *IEEE Journal of Solid-State Circuits,* Vol 38, Page(s): 319~328, Feb 2003.

[38] N. Azizi, A. Moshovos, and F. N. Najm, “Low-Leakage Asymmetric-Cell SRAM,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design,* Page(s): 48~51, Aug 2002.

[39] S. Heo, et al., “Dynamic Fine-Grain Leakage Reduction using Leakage-Biased Bitlines,” *IEEE/ACM Int. Symp. on Computer Architecture*, Page(s): 137~147, May 2002.

[40] Z. Hu, et al., “Managing Leakage for Transient Data: Decay and Quasi-Static 4T Memory Cells,” *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 52~55, Aug 2002.

[41] S. Yang, et al., “An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-caches,” *IEEE/ACM Int. Symp. High-Performance Computer Architecture*, Page(s): 147~157, Feb 2001.

[42] M. Powell, et al., “Reducing Leakage in a High-Performance Deep-Submicron Instruction Cache,” *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol 9, Page(s): 77~89, Feb 2001.

[43] S. Yang, et al., “Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay,” *IEEE/ACM Int. Symp. on High-Performance Computer Architecture*, Page(s): 134~144, Feb 2002.

[44] S. Kaxiras, et al., “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,” *IEEE/ACM Int. Symp. on Computer Architecture*, Page(s): 240~251, Jun 2001.

[45] H. Zhou, et al., “Adaptive Mode-Control: A Static-Power-Efficient Cache Design”, *IEEE/ACM Int Conf. on Parallel Architectures and Compilation Techniques*, Page(s): 61~70, Sep 2001.

[46] H. Hanson, et al., “Static Energy Reduction Techniques for Microprocessor Caches”, *IEEE Int. Conf. on Computer Design*, Page(s): 276~283, Sep 2001.

[47] H. Hanson, et al., “Static Energy Reduction Techniques for Microprocessor Caches,” *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol 11, Page(s): 303~313, Jun 2001.

[48] S. Kim, et al., “Predictive Precharging for Bitline Leakage Energy Reduction,” *IEEE the 15th ASIC/SOC Conf.*, Page(s): 36~40, Sep 2002.

[49] S. Yang and B. Falsafi, “Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches,” *IEEE/ACM the 36th Int. Symp. on Microarchitecture*, Page(s): 67~78, Dec 2003.

[50] W. Zhang, et al., “Compiler-Directed Instruction Cache Leakage Optimization,” *IEEE/ACM 35th Int. Symp. on Microarchitecture*, Page(s): 208~218, Nov 2002.

[51] W. Zhang, et al., “A Compiler Approach for Reducing Data Cache Energy,” *the 17th Int. Conf. on Supercomputing*, Page(s): 76~85, Jun 2003.

[52] J. Rabaey, A. Chandrakasan, and B. Nikolic, “Digital Integrated Circuits: A Design Perspective — 2nd ed,” Prentice-Hall, 2002.

[53] S. Wolf, “Silicon Processing for The VLSI Era — The Submicron MOSFET,” Lattice Press, Vol 3, Page(s): 213~222, 1995.

[54] A. Keshavarzi, K. Roy, and C. Hawkins, “Intrinsic Leakage in Low Power Deep Submicron CMOS ICs,” *IEEE Int. Test Conf.*, Page(s): 146~155, Nov 1997.

[55] R. Ho, K. Mai, and M. Horowitz, “The Future of Wires,” *IEEE Proceedings*, Vol 89, Page(s): 490~504, Apr 2001.

[56] Semiconductor Industry Association, http://public.itrs.net/Files/2001ITRS/Links/design/FO4writeup.pdf.

[57] M. Hrishikesh et al., “The Optimal Logic Depth per Pipeline Stage is 6 to 8 FO4 Inverter Delays,” *IEEE/ACM Int. Symp. on Computer Architecture*, Page(s): 14~24, May 2002.

[58] X. Zhang, “Coupling Effects on Wire Delay. Challenges in Deep Submicron VLSI Design,” *IEEE Circuits and Devices Magazine*, Vol 12, Page(s): 12~18, Nov 1996.

[59] T. May and M. Woods, “Alpha- Particle Induced Soft Errors in Dynamic Memories,” *IEEE Transactions on Electronic Devices*, Vol 26, Page(s): 2~9, Jan 1979.

[60] P. Hazucha, C. Svensson, “Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate,” *IEEE Transactions on Nuclear Science*, Vol 47, Page(s): 2586~2594, Dec 2000.

[61] S. Wilton and N. Jouppi, “An Enhanced Access and Cycle Time Model for On-Chip Caches,” *Western Research Laboratory Research Report 93/5*, Jul 1993.

[62] A. Smith, “Cache Memories,” *Computing Surveys*, Vol 14, Page(s): 473~530, Sep 1982.

[63] J. Smith and W.-C. Hsu, “Prefetching in Supercomputer Instruction Caches,” *Proc of Int. Conf. on Supercomputing*, Page(s): 588~ 597 Nov 1992.

[64] P.Y.-T. Hsu, “Designing the TFP microprocessor,” *IEEE Micro*, Vol 14, Page(s): 23~33, Apr 1994.

[65] J. Pierce and T. Mudge, “Wrong-path Instruction Prefetching,” IEEE/ACM *Proc. Int. Symp. on Microarchitecture*, Page(s): 165~175, Dec 1996.

[66] G. Reinman, B, Calder, and T. Austin, “Fetch Directed Instruction Prefetching,” *Proc. Int. Symp. on Microarchitecture*, Page(s): 16~27, Nov 1999.

[67] http://www.originlab.com/

[68] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: An Infrastructure for Computer System Modeling,” *IEEE Computer*, Vol. 35, Page(s): 59~67, Feb 2002.

[69] J. Hennessy, et al., “Computer architecture — A Quantitative Approach, 3rd

ed.," *Morgan Kaufmann,* Page(s): 406~408, 2003.

[70] Standard Performance Evaluation Corporation, http://www.specbench.org.

[71] T. Sherwood, et al., "Automatically Characterizing Large Scale Program Behavior," *ACM Int Conf. on Architectural Support for Programming Languages and Operating Systems,* Page(s): 45~47, Oct 2002.

[72] Rambus Inc., "800/1066MHz RDRAM Advanced Information," http://www.rambus.com, Ver. 0.6, 2002.

[73] V. Delaluz, et al., "Compiler-Directed Array Interleaving for Reducing Energy in Multi-Bank Memories," *IEEE Asia South Pacific Design Automation Conf.*, Page(s): 288~293, Jan 2002.

[74] T. Mudge, "Power: A First Class Design Constraint," *IEEE Computers*, Vol. 34, Page(s): 52~57, Apr 2001.

[75] J. Hu, et al., "Exploiting Program Hotspots and Code Sequentiality for Instruction Cache Leakage Management," *IEEE/ACM Int. Symp. on Lower Power Electronics and Design*, Page(s): 402~407, Aug 2003.

[76] N. S. Kim, et al., "Drowsy Instruction Caches – Reducing Leakage Power using Dynamic Voltage Scaling and Cache Sub-bank Prediction," *IEEE/ACM Int. Symp. on Microarchitecture*, Page(s): 219~230, Nov. 2002.

[77] N. S. Kim, et al., "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power," *IEEE Transactions on VLSI*, Vol 12, Page(s): 167~184, Feb 2004.

[78] T. May and M. Woods, "Alpha-Particle-Induced Soft Errors in Dynamic Memories," *IEEE Transaction on Electron Devices*, Vol 26, Jan 1979.

[79] A. Chandrakasan, W. Bowhill, and F. Fox, "Design of High-Performance Microprocessor Circuit," *IEEE Press*, Pages(s): 98~115, 2001.

[80] N. S. Kim, et al., "Leakage Current — Moore's Law Meets Static Power," *IEEE Computer*, Page(s): 68~75, Dec. 2003.