5-1-2000

Dynamically Resizable Instruction Cache: An Energy-Efficient and High-Performance Deep-Submicron Instruction Cache

Se-Hyun Yang
Purdue University School of Electrical and Computer Engineering

Michael Powell
Purdue University School of Electrical and Computer Engineering

Babak Falsafi
Purdue University School of Electrical and Computer Engineering

Kaushik Roy
Purdue University School of Electrical and Computer Engineering

T. N. Vijaykumar
Purdue University School of Electrical and Computer Engineering

DYNAMICALLY RESIZABLE INSTRUCTION CACHE: AN ENERGY-EFFICIENT AND HIGH-PERFORMANCE DEEP-SUBMICRON INSTRUCTION CACHE

SE-HYUN YANG
MICHAEL POWELL
BABAK FALSAFI
KAUSHIK ROY
T. N. VIJAYKUMAR

TR-ECE 00-7
MAY 2000

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Dynamically Resizable Instruction Cache: An Energy-Efficient and High-Performance Deep-Submicron Instruction Cache

Se-Hyun Yang, Michael Powell, Babak Falsafi, Kaushik Roy, and T.N. Vijaykumar

Purdue ICALP Project
School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285
icalp@ecn.purdue.edu, http://www.ece.purdue.edu/~icalp

Table of Contents

List of Tables

List of Figures

Abstract

Increasing levels of on-chip integration have enabled steady improvements in modem microprocessor performance, but have also resulted in high energy dissipation. Deep-submicron CMOS designs maintain high transistor switching speeds by scaling down the supply voltage and proportionately reducing the transistor threshold voltage. Lowering the threshold voltage increases leakage energy dissipation due to an exponential increase in the leakage current flowing through a transistor even when the transistor is not switching. Estimates from the VLSI circuit community suggest a five-fold increase in leakage energy dissipation in every future generation. Modem microarchitectures aggravate the leakage energy problem by investing vast resources in on-chip cache hierarchies because leakage energy grows with the number of transistors. While demand on cache hierarchies varies both within and across applications, modem caches are designed to meet the worst-case application demand, resulting in poor utilization of on-chip caches, which in turn leads to energy inefficiency.

This paper explores an integrated architectural and circuit-level approach to reduce leakage energy dissipation in instruction caches (i-caches) while maintaining high performance. Using a simple adaptive scheme, we exploit the variability in application demand in a novel cache design, the Dynamically Resizable i-cache (DRI i-cache), by dynamically resizing the cache to the size required at any point in application execution. At the circuit-level, the DRI i-cache employs a novel mechanism, called gated-V_{dd}, which effectively turns off the supply voltage to the SRAM cells in the DRI i-cache's unused sections, virtually eliminating leakage in these sections. Ow adaptive scheme gives DRI i-caches tight control over the number of extra misses caused by resizing, enabling the DRI i-cache to contain both performance degradation and extra energy dissipation due to increased number of accesses to lower cache levels. simulations using the SPEC95 benchmarks show that a 64K DRI i-cache reduces, on average, both the leakage energy-delay product.and average size by 62%, with less than 4% impact on execution time.

1 Introduction

The ever-increasing levels of on-chip integration in the recent decade have enabled phenomenal increases in computer system performance. Unfortunately, the performance improvement has been also accompanied by an increase in chips' power and energy dissipation. Higher power and energy dissipation require more expensive packaging and cooling technology, increase cost, and decrease reliability of products in all segments of computing market from portable systems to high-end servers [24]. Moreover, higher power and energy dissipation significantly reduce battery life and diminish the utility of portable systems.

Historically, the primary source of energy dissipation in CMOS transistor devices has been the ***dynamic energy*** due to charging/discharging load capacitances when the device switches. Chip designers have relied on scaling down the transistor supply voltage in subsequent generations to reduce the dynamic energy dissipation due to a much larger number of on-chip transistors.

Maintaining high transistor switching speeds, however, requires a commensurate down-scaling of the transistor threshold voltage along with the supply voltage [22]. The International Technology Roadmap for Semiconductors [23] predicts a steady scaling of supply voltage with a corresponding decrease in transistor threshold voltage to maintain a 30% improvement in performance every generation. Transistor threshold scaling, in turn, gives rise to a significant amount of ***leakage energy*** dissipation due to an exponential increase in leakage current even when the transistor is not switching [7,32,28,20,26,14,11]. Borkar [7] estimates a factor of 7.5 increase in leakage current and a five-fold increase in total leakage energy dissipation in every chip generation.

State-of-the-art microprocessor designs devote a large fraction of the chip area to memory structures—e.g., multiple levels of instruction (i-cache) caches and data (d-cache) caches, TLBs, and prediction tables. For instance, 30% of Alpha 21264 and 60% of StrongARM are devoted to cache and memory structures [18]. Unlike dynamic energy which depends on the number of actively switching transistors, leakage energy is a function of the number of on-chip transistors, independent of their switching activity. As such, caches account for a large (if not dominant) component of leakage energy dissipation in recent designs, and will continue to do so in the future. Recent energy estimates for 0.13μ processes indicate that leakage energy accounts for in excess of 50% of the total energy dissipated in cache memories [6]. Unfortunately, current proposals for energy-efficient cache architectures [16,5,2] only target reducing dynamic energy and do not impact leakage energy.

There are a myriad of circuit techniques to reduce leakage energy dissipation in transistors/circuits (e.g., multi-threshold [30,26,20] or multi-supply [12,27] voltage design, dynamic threshold [29] or dynamic supply [9] voltage design, transistor stacking [32], and cooling [7]). These techniques, however, suffer from three significant shortcomings. First, they often impact circuit performance and are only applicable to circuit sections that are not performance-critical [13]. Second, they may require sophisticated fabrication process and increase cost (e.g., dynamic supply- and threshold-voltage designs). Finally, the circuit techniques apply low-level leakage energy reduction at ***all times*** without taking into account the application behavior and the dynamic utilization of the circuits.

Current high-performance microprocessor designs incorporate multi-level cache hierarchies on chip to reduce off-chip access frequency and improve performance. Modem cache hierarchies are designed to satisfy the demands of the most memory-intensive applications or application phases. The actual utilization of caches, however, varies widely both ***within*** and ***across*** applications. Recent studies on block frame utilization in caches [21], for instance, show that at any given instance in an application's execution, on average over half of the block frames are "dead — i.e., they miss upon a subsequent reference. These "dead block frames continue dissipating leakage energy while not holding useful data.

This paper presents the first integrated architectural and circuit-level approach to reduce leakage energy dissipation in deep-submicron cache memories. We propose a novel instruction cache (i-cache) design, the

Dynamically ResIzable instruction cache (DRI i-cache), which dynamically resizes itself to the size required at any point during application execution and virtually turns off the supply voltage to the rest of the cache's unused sections to eliminate leakage. At the architectural level, a DRI i-cache relies on simple but effective techniques to exploit the variability in i-cache usage and reduce the i-cache size dynamically to capture the application's primary instruction working set. At the circuit-level, a DM i-cache uses a recently-proposed mechanism, *gated-V_{dd}* [1], which reduces leakage by effectively turning off the supply voltage to the SRAM cells of the cache's unused block frames.

Using state-of-the-art cycle-accurate architectural simulation and energy estimation circuit tools, we show the following.

- There is a large variability in L1 i-cache utilization both ***within*** and ***across*** applications. Using a simple adaptive hardware scheme, a DRI i-cache effectively exploits the variability by dynamically resizing the cache to accurately fit the application's working set. Simulations using SPEC applications indicate that a DRI i-cache reduces the average size of a 64K-cache by 62% with performance degradation constrained within 4%, and by 78% with higher performance degradation.
- Previous resizing techniques coarsely vary associativity without controlling the extra misses incurred due to resizing, resulting in performance degradation and extra energy dissipation to access lower cache levels [2]. Our adaptive scheme gives DRI i-caches tight control over the number of extra misses by constraining the miss rate to stay close to a preset value, enabling the DM i-cache to contain both performance degradation and the extra lower-cache-level energy dissipation.
- A DRI i-cache effectively integrates architectural and the gated-V_{dd} circuit techniques to reduce leakage in an L1 i-cache. Compared to a conventional i-cache, a DM i-cache reduces the leakage energy-delay product by 62% with performance degradation constrained within 4%, and by 67% with higher performance degradation.
- Because higher set-associativities encourage more downsizing, and larger sizes imply larger relative size reduction, DRI i-caches achieve even better energy-delay products with higher set-associativity and larger size.
- Our adaptive scheme is robust in that it is not sensitive to many of the adaptivity parameters, and performs predictably without drastic reactions to varying the rest of the adaptivity parameters.

The rest of the paper is organized as follows. In Section 2, we describe the architectural techniques to resize i-caches dynamically. In Section 3, we describe the gated-V_{dd} circuit-level mechanism to reduce leakage in SRAM cells. In Section 4, we describe our experimental methodology. In Section 5, we present experimental results. Section 6 and Section 7 present the related work and conclusions, respectively.

2 DRI I-Cache: Reducing the Leakage in Deep-Submicron I-Caches

This paper proposes the ***Dynamically Reslzable instruction cache (DRI i-cache).*** The key observation behind a DRI i-cache is that there is a large variability in i-cache utilization both ***within*** and ***across*** programs leading to large energy inefficiency for conventional caches in deep-submicron designs; while the memory cells in a cache's unused sections are not actively referenced, they leak current and dissipate energy. A DRI i-cache's novelty is that it dynamically estimates and adapts to the required i-cache size, and uses a novel circuit-level technique, gated V_{dd} [1], to turn off the supply voltage to the cache's unused memory cells. In this section, we will describe the anatomy of a DRI i-cache. In the next section, we will present the circuit technique to gate a memory cell's supply voltage.

The large variability in i-cache utilization is inherent to an application's execution. Application programs often break down the computation into distinct phases. In each phase, an application typically iterates and computes over a set of data. The code size executed in each phase dictates the required i-cache size for that phase. Our ultimate goal is to exploit the variability in the code size and the required i-cache size across application phases to save energy. The key to our leakage energy saving technique is to have a minimal impact on performance and a minimal increase in dynamic energy dissipation.

To exploit the variability in icache utilization, hardware (or software) must provide accurate mechanisms to determine a transition among two application phases and estimate the required new i-cache size. Inaccurate cache resizing may significantly increase the access frequency to lower cache levels, increase the dynamic energy dissipated, and degrade performance, offsetting the gains from leakage energy savings. Resizing may also affect block placement in the cache requiring existing blocks to be moved to new frames, incurring overhead. A mechanism is also required to determine how long an application phase executes so as to select phases that have long enough execution times to amortize the resizing overhead.

In this paper, we use a simple and intuitive all-hardware design to resize an i-cache dynamically. Our approach to cache resizing increases or decreases the number of active cache sets. Alternatively, we could increase/decrease associativity, as is proposed for reducing dynamic energy in [2]. This alternative, however, has several key shortcomings. First, it assumes that we start with a base set-associative cache and is not applicable to direct-mapped caches, which are widely used due to their access latency advantages. Second, changing associativity is a coarse-grained approach to resizing and may increase both capacity and conflict miss rates in the cache. Such an approach increases the cache resizing overhead, significantly reducing the opportunity for energy reduction.

While many of the ideas in this paper apply to both i-caches and dcaches, we focus on i-cache designs in this paper. Our approach to resizing caches requires that upon downsizing, the cache's unused sections be turned off to save energy. Because cache blocks in a d-cache may be modified, dynamically resizing d-caches requires that either the modified data in the cache's unused sections be written back., potentially offsetting the gains from saving energy [2] and incurring high writeback latency, or the modified cache blocks remain "on". The latter complicates the cache design (Section 2.2) because accesses to the cache require lookups in both the "on" and "off" sections of the cache, incurring prohibitively high latency upon lookup. This paper is the first step towards designing dynamically resizable caches, and as such we focus on i-caches. Studying d-cache designs is beyond the scope of this paper.

In the rest of this section, we will first describe the mechanisms to detect phase transitions and cache resizing and discuss cache lookup and placement strategies for a DRI i-cache. Next, we will discuss the hardware and software implications for our new design. Finally, we present the impact on dynamic energy dissipation using our design.

2.1 Basic DRI I-Cache Design

Much like conventional adaptive computing frameworks, our cache uses a set of parameters to monitor, react, and adapt to changes in application behavior and system requirements dynamically. A DRI i-cache divides an application's execution time into fixed-length intervals (measured in the number of instructions executed) to monitor the cache's performance, and decides at the end of every interval if a change in cache size is necessary. We use miss rate as the primary metric for monitoring the cache's performance. The keys to a successful DRI i-cache design are mechanisms to control the cache size accurately while preventing a significant increase in the cache miss rate. A large miss rate increase may both prohibitively increase execution time and the dynamic energy dissipated in the lower level caches, offsetting the leakage energy savings. Therefore, the key parameters in our design are those that directly control the cache's miss rate.

Figure 1 depicts the anatomy of a direct-mapped DRI i-cache (the same design applies to set-associative caches). The adaptive mechanism monitors the cache in fixed-length intervals, the sense interval, measured in number of dynamic instructions (e.g., 1 million instructions). A miss counter counts the number of DRI i-cache misses in each sense interval. At the end of each sense interval, the cache upsizes/downsizes, depending on whether the miss counter is lower/higher than a preset value, the miss-bound. The factor by which the cache resizes (up or down) is called the divisibility. For example, divisibility of two halves the cache size upon every downsize and miss-bound set at 10,000 triggers a downsize if the cache incurs more than 10,000 misses in a sense interval. If, however, the i-cache size would get smaller than a preset size, the size-bound (e.g., 1 K), the cache does not downsize and stays at the same size.

All the cache parameters — i.e., the interval length, the size-bound, the miss-bound, and the divisibility — can be set either dynamically or statically. This paper is a first step towards understanding a resizable cache design. As such, we focus on designs that statically set the values for the parameters prior to the start of program execution.

Among these parameters, the key parameters that control the i-cache's size and performance are the miss-bound and size-bound. The combination of these two key parameters provides accurate and tight control over the cache's performance. Miss-bound allows the cache to react and adapt to an application's instruction working set by "bounding" the cache's miss rate in each monitoring interval. Thus, the miss-bound provides a "fine-grain" resizing control between any two intervals independent of the cache size. Applications typically require a specific minimum cache capacity beyond which they incur a large number of capacity misses and thrash. Size-bound provides a "coarse-grain" resizing control by preventing the cache from thrashing by downsizing past a minimum size.

The other two parameters, the sense interval length and divisibility, are less-critical to DRI i-cache performance. Intuitively, the sense interval length allows selecting a sense interval length that best matches an application's phase transition times, and the divisibility determines the rate at which the i-cache is resized.

Resizing the cache requires that we dynamically change the cache block lookup and placement function. Conventional (direct-mapped or set-associative) i-caches use a fixed set of index bits from a memory reference to locate the set to which a block maps. Resizing the cache either reduces or increases the total number of cache sets thereby requiring a larger or smaller number of index bits to look up a set. Our design uses a mask to find the right number of index bits used for a given cache size (Figure 1). Every time the cache downsizes, the mask shifts to the right to use a smaller number of index bits and vice versa. Therefore, downsizing removes the highest-numbered sets in the cache in groups of powers of two. The mask can be folded into the address decoder trees of the data and tag arrays, so as to minimize the impact on the lookup time.

Because smaller caches use a small number of index bits, they require a larger number of tag bits to distinguish data in block frames. Because a DRI i-cache dynamically changes its size, it requires a different

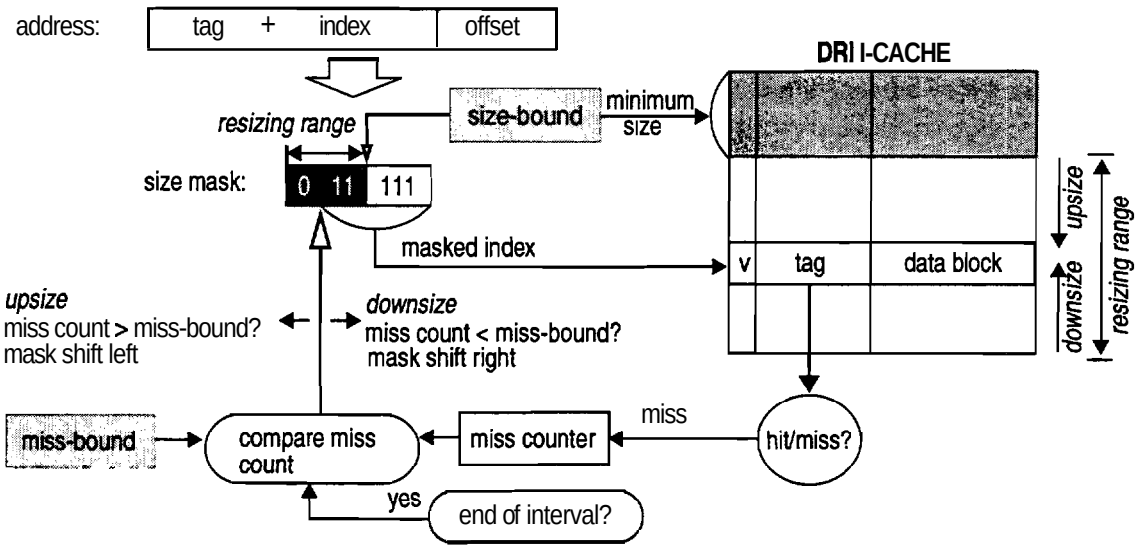

FIGURE 1: Anatomy of a DRI i-cache.

number of tag bits for each of the different sizes. To satisfy this requirement, our design maintains as many tag bits as required by the smallest size to which the cache may downsize itself. Thus, we maintain more tag bits than conventional caches of equal size. We define the extra tag bits to be ***the resizing tag bits.*** The size-bound dictates the smallest allowed size and, hence, the corresponding number of resizing bits. For instance, for a 64K DRI i-cache with a size-bound of 1K, the tag array uses 16 (regular) tag bits and 6 resizing tag bits for a total of 22 tag bits to support downsizing to 1K. The resizing tag bits increase the dynamic energy dissipated in the cache as compared to a conventional design. We will discuss the dynamic energy implications of resizing tag bits in Section 2.3 and present results in Section 5 that indicate a DRI i-cache has an overall minimal impact on the dynamic energy dissipated.

2.2 Implications on Block Lookups

Using the resizing tag bits, we ensure that the cache functions correctly at every individual size. However, transitioning from one size to another may still cause problems in cache lookup. Because resizing modifies the set-mapping function for blocks (by changing the index bits), it may result in an incorrect lookup if the cache contents are not moved to appropriate places or flushed before resizing. For instance, a 64K cache maintains only 16 tag bits whereas a 1K cache maintains 22 tag bits. As such, even though downsizing the cache from 64K to 1K allows the cache to maintain the upper 1K contents, the tags are not comparable (Figure 2 left). While a simple solution, flushing the cache or moving block frames to the appropriate places may incur prohibitively large amounts of overhead. Our design does not resort to this solution because we already maintain all the tag bits necessary for the smallest cache size at all times; i.e., a 64K cache maintains the same 22 tag bits from the block address that a 1K cache would. This way, a tag comparison can proceed independent of the cache size obviating the need to move the blocks or flush the cache after resizing.

Moreover, upsizing the cache may complicate lookup because blocks map to different sets in different cache sizes. Figure 2 (right) illustrates an example in which ***block*** A maps to a different set when upsizing the cache from 1K to 64K. Such a scenario creates two problems. A lookup for A after upsizing fails to find A, and therefore fetches and places A into a new set. While the overhead of such (compulsory) misses after upsizing may be negligible and can be amortized over the sense interval length, such an approach will result in multiple ***aliases*** of *A* in the cache. Unlike d-caches, however, in the common case a processor only

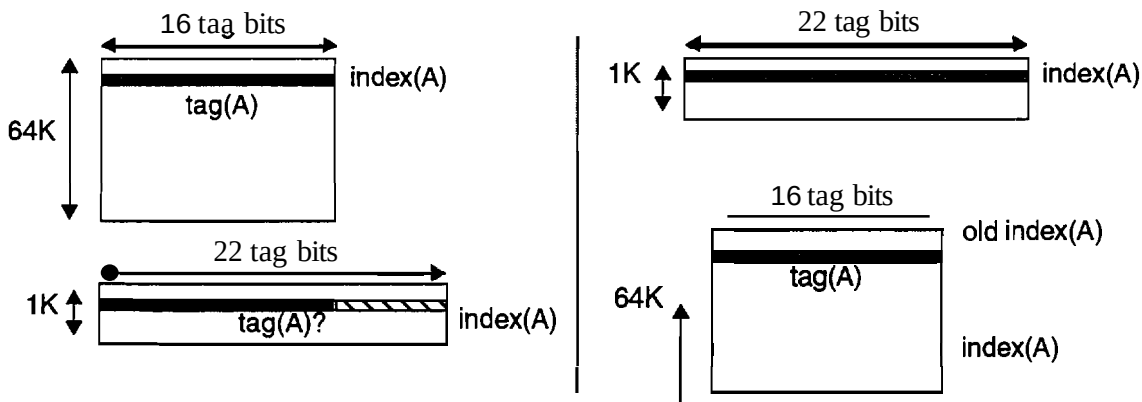

FIGURE 2: Block lookup and frame aliasing in a DRI i-cache.

reads and fetches instructions from an i-cache and does not modify a block's contents. Therefore, allowing multiple aliases does not interfere with processor lookups and instruction fetch in i-caches.

There are scenarios, however, which require invalidating all aliases of a block in the i-cache. Unmapping an instruction page (when swapping the page to the disk) requires invalidating all of the page's blocks in the i-cache. Similarly, dynamic libraries require call sites which are typically placed in the heap and require coherence between the icache and the d-cache. Conventional systems, however, often flush the icache or d-cache to maintain coherence between them because the above operations are infrequent. Moreover, these operations typically involve OS intervention and incur high overheads, amortizing the cache flush overhead.

2.3 Impact on Energy and Performance

Cache resizing helps reduce leakage energy by allowing a DRI i-cache to turn off the cache's unused sections. Resizing, however, may adversely impact the miss rate (as compared to a conventional i-cache) and the access frequency to the lower-level (L2) cache. The increase in L2 accesses may impact both execution time and the dynamic energy dissipated in L2. While the impact on execution time depends on an application's sensitivity to i-cache performance, the higher miss rate may significantly impact the dynamic energy dissipated due to the growing size of on-chip L2 caches [2]. A DRI i-cache may also increase the dynamic energy dissipated as compared to a conventional cache due to the extra resizing tag bits in the tag RAM. The combined effect of the above may offset the gains in leakage energy. In this section, we qualitatively analyze the impact on performance and energy dissipation of a DRI i-cache. In Section 5.3, we present simulation results which indicate that a DRI i-cache can significantly reduce leakage energy with minimal impact on execution time and dynamic energy.

There are two sources of increase in the miss rate when resizing. First, resizing may require remapping of data into the cache and incur a large number of (compulsory) misses at the beginning of a sense interval. The resizing overhead is dependent on both the resizing frequency and the sense interval length. Fortunately, applications tend to have at most a small number of well-defined phase boundaries at which the i-cache size requirements drastically change due to a change in the instruction working set size. Our results, however, indicate that optimal interval lengths to match application phase transition times are long enough to help amortize the overhead of moving blocks around at the beginning of an interval (Section 5.3).

Second, downsizing may be suboptimal and result in a significant increase in the miss rate when the required cache size is slightly below a given size. Such a scenario will lead to both high miss rates for intervals in which an application's working set does not fit in the cache, and frequent unnecessary switching between two cache sizes. The impact on the miss rate is highest at very small cache sizes when the cache begins to thrash. Much as other adaptive systems, a DRI i-cache incorporates a simple throttling mecha-

Resizing	Parameters	Leakage	M i rate	L1 Tag Energy
aggressive	miss-bound >> conventional miss rate size-bound low	-1	↑	↑
conservative	miss-bound - conventional miss rate size-bound high	↑	↓	↓

Table 1: Impact of resizing on leakage, miss rate, and L1 dynamic energy.

nism (using a 3-bit saturating counter with resizing backoff) to prevent the system from unnecessary resizings. Moreover, the size-bound ***guarantees*** that the cache never resizes beyond a given size, preventing the cache from thrashing.

Miss-bound and size-bound control a DRI i-cache's aggressiveness in reducing the cache size and leakage energy. Table 1 depicts the impact of resizing on performance and energy. In an aggressive DRI i-cache configuration with a large miss-bound and a small size-bound, the cache is allowed to resize more often and to small cache sizes, thereby aggressively reducing leakage. Small cache sizes, however, may lead to high miss rates, a high increase in dynamic energy dissipated in L2, and a high increase in dynamic energy dissipated in L1 due to a large number of resizing tag bits. Depending on an application's performance sensitivity to i-cache miss rates, the higher miss rates may increase execution time, indirectly increasing the overall energy dissipated due to a higher execution time.

A conservative DRI i-cache configuration maintains a miss rate which is close to the miss rate of a conventional icache of the same base size, and bounds the downsizing to higher cache sizes so as to prevent thrashing and significantly increasing the miss rate. Such a configuration reduces leakage with minimal impact on execution time and dynamic energy dissipated.

The adaptivity parameters, sense interval length and divisibility, may also affect a DRI i-cache's ability to adapt to the required i-cache size accurately and timely. While a larger divisibility favors applications with drastic changes in icache requirements, it makes size transitions more coarse reducing a DRI icache's opportunity to adapt to a size closest to the required size. Similarly, while longer sense intervals may span over multiple application phases reducing opportunity for resizing, shorter intervals may result in a high resizing overhead due to a large number of compulsory misses after resizing. Our results, however, indicate that the sense interval length and divisibility are less critical than the miss-bound and size-bound to controlling the extra misses in a DRI i-cache (Section 5.6).

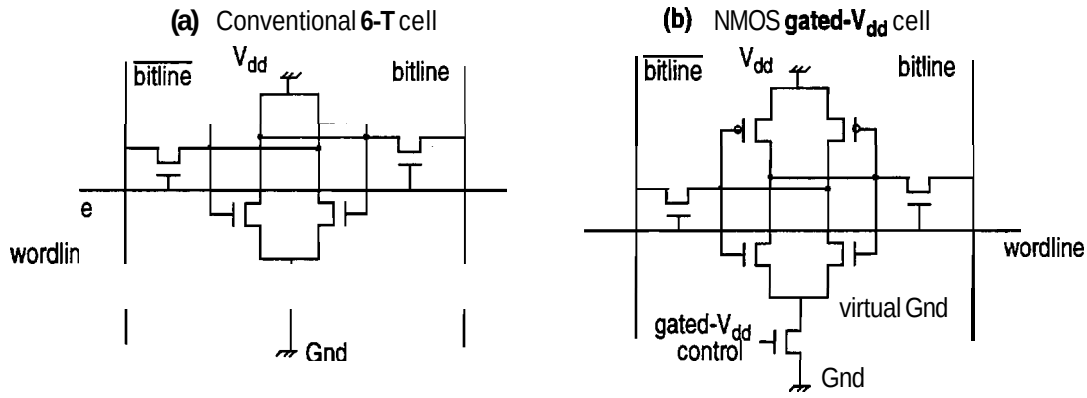

FIGURE 3: 6-T SRAM cell schematics: (a) conventional, (b) with NMOS gated-V_{dd}.

3 Gated-V_{dd}: A Circuit-level Mechanism for Supply-Voltage Gating

Current technology scaling trends [7] require aggressive scaling down of the threshold voltage (V,) to maintain transistor switching speeds. At low V„ however, there is a subthreshold leakage current through transistors, even when operating in the "cut-off' region, i.e., when the transistor is "off' [22]. The leakage current increases exponentially with decreasing threshold voltage, resulting in a significant amount of leakage energy dissipation at low V„.

To prevent the leakage energy dissipation in a DRI i-cache from limiting aggressive threshold-voltage sealing, we use a circuit-level mechanism called *gated-V_{dd}* [1]. Gated-V_{dd} enables a DRI i-cache to turn off effectively the supply voltage and eliminate virtually all the leakage energy dissipation in the cache's unused sections. The key idea is to introduce an extra transistor in the leakage path from the supply voltage (V_{dd}) to the ground (Gnd) of the cache's SRAM cells; the extra transistor is turned on in the used and turned off in the unused sections, essentially "gating" the cell's supply voltage. Gated-V_{dd} maintains the performance advantages of lower supply and threshold voltages while reducing the leakage.

The fundamental reason for gated-V_{dd} achieving exponentially lower leakage is that two "off' transistors connected in series incur an order of magnitude lower leakage; this effect is due to self reverse-biasing of the stacked transistors, and is called the stacking *effect* [32]. Gated-V_{dd}'s extra transistor connected in series with the SRAM cell transistors produces the stacking effect when the gated-V_{dd} transistor is turned off, resulting in a high reduction in leakage. When the gated-V_{dd} transistor is turned "on", the cell is said to be in "active" mode and when turned "off', the cell is said to be in "standby" mode. In the interests of limiting the number of terms defined in the paper, we will continue to use "on" and "off' modes.

Figure 3 (a) depicts the anatomy of a conventional 6-T SRAM cell with dual-bitline architecture we assume in this paper. On a cache access, the corresponding row's wordline is activated by the address decode logic, causing the cells to read their values out to the precharged bitlines or to write the values from the bitlines into the cells. The two inverters in Figure 3 (a) each have a V_{dd} to Gnd leakage path going through an NMOS or a PMOS transistor connected in series. Depending on the bit value (of 0 or 1) held in the cell, the PMOS transistor of one and the corresponding NMOS transistor of the other inverter are "off'. Figure 3 (b) shows a DRI i-cache SRAM cell using an NMOS gated-V_{dd} transistor. When the gated-V_{dd} transistor is "off', it is in series with the "off' transistors of the inverters, producing the stacking effect. The DRI i-cache resizing circuitry keeps the gated-V_{dd} transistors of the used sections turned on and the unused sections turned off.

Much as conventional gating techniques, the gated-V_{dd} transistor can be shared among multiple circuit blocks to amortize the overhead of the extra transistor. For example, in a DRI i-cache, a single gated-V_{dd} transistor can be shared among the SRAM cells of one or more cache blocks. To reduce the impact on SRAM cell speed, the gated-V_{dd} transistor must be carefully sized with respect to the SRAM cell transistors it is gating. While the gated-V_{dd} transistor must be made large enough to sink the current flowing through the SRAM cells during a read/write operation in the "on" mode, too large a gated-V_{dd} transistor may reduce the stacking effect, thereby diminishing the power savings. Moreover, large: transistors also increase the area of overhead due to gating.

Gated-V_{dd} may be implemented using either an NMOS transistor connected between the SRAM cell and Gnd or a PMOS transistor connected between V_{dd} and the cell. Using a PMOS or an NMOS gated-V_{dd} transistor presents a trade-off between area overhead, leakage reduction, and impact on performance [1]. Moreover, gated-V_{dd} can be coupled with a dual-threshold voltage (dual-V,) process technology [28] to achieve even larger reductions in leakage. Dual-V, technology allows integrating transistors with two different threshold voltages. With dual-V, technology, the SRAM cells use low-V, transistors to maintain high speed and the gated-V_{dd} transistors use high-V_t to achieve additional leakage reduction. In this paper, we assume NMOS gated-V_{dd} transistors with dual-Vt for optimal performance, energy, and area results [1].

Processor Parameters	
Instruction issue & decode bandwidth	8 issues per cycle
L1 i-cache/ L1 DRI i-cache	64K, direct-mapped 1 cycle latency
L1 d-cache	64K, 2-way (LRU) 1 cycle latency
L2 cache	1M, 4-way, unified 12 cycle latency
Memory access latency	80 cycles + 4cycles per 8 bytes
Reorder buffer size	128
LSQ size	128
Branch predictor	2-level hybrid

Table 2: Processor configuration parameters.

name	input	#of inst(billions)
Integer benchmarks		
compress	test	0.035
gcc	train	1.276
go	train	0.955
ijpeg	test	0.553
li	train	0.183
m88ksim	test	0.490
perl	train	0.040
Floating point benchmarks		
applu	train	0.288
apsi	train	2.349
fpppp	train	0.331
hydro2d	test	0.974
mgrid	test	4.422
su2cor	test	1.054
swim	test	0.849
tomcatv	test	2.798

Table 3: Applications and input sets.

4 Methodology

We use Simplescalar-2.0 [10] to simulate an L1 DRI i-cache in the context of an out-of-order microprocessor. Table 2 shows the base configuration for the simulated system. Table 3 presents the benchmarks we use in this study, the corresponding input data sets, and the number of dynamic instructions executed. We run all of SPEC95 with the exception of two floating-point benchmarks and one integer benchmark (in the interests of reduced simulation turnaround time).

To determine the energy usage of a DRI i-cache, we use geometry and layout information from CACTI [31]. Using spice information from CACTI to model the 0.18μ SRAM cells and related capacitances, we determine the leakage energy of a single SRAM cell and the dynamic energy of read and write activities on single rows and columns. This information is used to determine energy dissipation for appropriate cache configurations. Area estimates are from a Mentor Graphics IC-Station layout of a single cache line.

All simulations use a power supply of 1.0 volt. We estimate cell access time and energy dissipation using Hspice transient analog analysis. The worst case read time is the time to lower the bitline to 75% of V_{dd} after the wordline is asserted. We compute "off' and "on" mode energy dissipation by measuring average energy dissipated by a steady state cache cells with the gated-V_{dd} transistor in the correct mode. We ensure that the SRAM cells are all initialized to a stable state before measuring read time or "on" mode leakage energy.

Technique	Cell V_t (V)	"On" Cell-Leakage per Cycle (nJ)	"Off' Cell-LeakageEnergy per Cycle (nJ)	Energy Savings (%)	Relative Read Time	Area Increase (%)
no gated-V_{dd}, high V_t	0.30	50	NIA	N/A	2.22	N/A
no gated-V_{dd}, low V_t	0.20	1740	N/A	N/A	1.00	N/A
gated-V_{dd}, low V_t	0.20	1740	53	97	1.01	3%

Table 4: Leakage Energy, read time, and area for gated-V_{dd}.

5 Results

In this section, we present experimental results on the performance and energy trade-off of a **DRI** i-cache, compared to a conventional i-cache. First we briefly present circuit results indicating the energy savings of gated-V_{dd}. Then we present energy savings achieved for the benchmarks, demonstrating a DRI i-cache's effectiveness at reducing average cache size and energy dissipation. Then we present the impact of **DRI** i-cache parameters on dynamic energy dissipation: effect of the number of extra tag bits on L1 dynamic energy, and effect of the miss-bound on **L2** dynamic energy. Next we show performance and energy results while varying DRI i-cache size and associativity. Finally, we present the effect of varying the **DRI** i-cache adaptivity parameters: divisibility and sense interval length.

5.1 Circuit Results

As transistor threshold voltages decrease, we expect read time to improve but leakage energy to increase drastically. In this section we will show that gated-V_{dd} virtually eliminates the leakage while maintaining fast read times. Table **4** shows leakage energy per cycle, relative read time, and the area overhead associated with gated-V_{dd}. We assume a dual-V, technology, with a **0.4** V V_t for the gated-V_{dd} transistor, and a **0.2** V V, for the **SRAM** cell transistors.

The "on" Leakage Cell Energy and "off' Leakage Cell Energy columns indicates leakage energy dissipated per cycle when the cell is in "on" and "off' mode, respectively. From the first two rows, we see that lowering the cache V_t from **0.3** to **0.2** V reduces the read time by over half but increases the leakage energy by more than a factor of **30.** From the third row we see that using gated-V_{dd}, the leakage energy can be reduced by 97% in "off' mode, confining the leakage to high-V, levels while maintaining low-V, speeds. This large reduction in leakage is key to ensuring that unused sections of the cache put in "off' mode dissipate little leakage energy.

If a gated-V_{dd} transistor is shared among an entire cache block's **SRAM** cells, the transistor needs to be wide enough to sink the current through the block's **SRAM** cells to prevent excessive degradation in read time. The width of the gated-V_{dd} transistor needs to be equivalent to the maximum number of cell transistors that can simultaneously switch in the cache block. Figure **4** shows a layout of 64 **SRAM** cells on the left and an adjoining **NMOS** gated-V_{dd} transistor connected to them. By constructing the gated-V_{dd} transistor such that the transistor width expands along the length of the cache line, only the width of the array increases with the addition of such a transistor, without increasing the array's height. The total increase in data array area due to the addition of the gated-V_{dd} transistor is about **3%.**

FIGURE 4: Layout of 64 SRAM cells connected to a single gated-V_{dd} NMOS transistor.

5.2 Energy Calculations Illustrating Leakage and Dynamic Energy Trade-off

A DRI i-cache decreases leakage energy by gating Vdd to cache sections in "off' mode but increases both L1 dynamic energy due to the extra resizing tag bits and L2 dynamic energy due to extra L1 misses. To account for both the decrease in leakage energy and increase in dynamic energy, we compute the Effective Ll leakage energy using three components, the L1 leakage energy, extra L1 dynamic energy, and extra L2 dynamic energy as follows:

Effective L1 DRI i-cache leakage energy = L1 leakage energy + extra L1 dynamic energy + Extra L2 dynamic energy
L1 leakage energy = Leakage energy of "on" portion + Leakage energy of "off" portion
Leakage energy of 'on" portion = ('On" portion size as fraction) x (Leakage energy of conventional cache per cycle) x (cycles)
Leakage energy of "off" portion ~= 0
Extra L1 dynamic energy = (resizing tag bits) x (Dynamic energy of 1 bitline per L1 access) x (L1 accesses)
Extra L2 dynamic energy = (Dynamic energy per L2 access) x (extra L2 accesses)

We now explain the equations. The Effective L1 leakage energy is the effective leakage energy dissipated by the DRI i-cache during the course of the application execution. The first component, the L1 leakage energy, is the leakage energy of the "on" and "off" portions of the DRI i-cache dissipated during the execution. We compute DRI i-cache's "on" portion leakage energy as the leakage energy dissipated by a conventional i-cache in one cycle times the DRI i-cache "on" portion size expressed as a fraction of the total size times the number of cycles. We obtain the average "on" portion size and the number of cycles from Simplescalar simulations. Using the low-V_t "on" Cell-Leakage energy in Table 4, we compute the leakage energy for a conventional i-cache. Because the "off' mode energy is a factor of 30 smaller than the "on" mode energy in Table 4, we set the "off' mode term to zero.

The second component is the extra L1 dynamic energy dissipated due to the extra resizing tag bits during the application execution. We compute this component as the number of resizing tag bits used by the program times the dynamic energy dissipated in one access of one resizing tag bitline in the L1 cache times the number of L1 accesses made in the program. The third component is the extra L2 dynamic energy dissipated in accessing the L2 cache due to the extra L1 misses during the application execution. We compute this component as the dynamic energy dissipated in one access of the L2 cache times the number of extra L2 accesses. To estimate the dynamic energy of one access of one resizing tag bitline and of one L2 access, we modify the Spice files supplied by CACTI and use the calculations for cache access energy in [15]. We tabulate the results of our calculations in Table 5, which simplifies the energy expressions as follows:

Effective L1 DRI i-cache leakage energy = L1 leakage energy + L1 extra dynamic energy + L2 extra dynamic energy

L1 leakage energy = ("On" portion size as fraction) x 0.91 x (cycles)
Extra L1 dynamic energy = (resizing tag bits) x 0.0022 x (L1 accesses)
Extra L2 dynamic energy = 3.6 x (extra L2 accesses)
Energy savings = Conventional cache leakage - Effective L1 DRI i-cache leakage energy

If the extra L1 and L2 dynamic energy components do not significantly add to L1 leakage energy, a DRI i-cache's energy savings will not be outweighed by extra (L1+L2) dynamic energy, as forecasted in

Energy component	Energy (nJ)
Dynamic energy per L2 access	3.6
Dynamic energy of 1 resizing tag bit per L1 access	0.0022
Conventional L1 cache leakage energy per cycle	0.91
Ratio of L1 leakage energyper cycle to L2 dynamic energy per access	0.25

Table 5: Cache energy components.

Section 2.3. To demonstrate that the components do not significantly add to L1 leakage energy, we compare each of the components to the L1 leakage energy and show that the components are much smaller than the leakage energy.

(Extra L1 dynamic energy)/ (L1 leakage energy)= [(resizing bits) x 0.0022 x (L1 accesses)]/ [('On—action) x 0.91 x (cycles)]
~ [(resizing bits) x 0.0022] / [('On' fraction) x 0.911
- 0.024 (if resizing == 5, "on" fraction == 0.25)

We compare the extra L1 dynamic energy against the L1 leakage energy by computing their ratio. We simplify the ratio by approximating the number of L1 accesses to be equal to the number of cycles (i.e., an L1 access is made every cycle L1 accesses and cycles), and cancelling the two in the ratio. If the number of resizing tag bits is 5 (i.e., the size-bound is a factor of 32 smaller than the original size), and the "on" portion is as small as half the original size, the ratio reduces to 0.024, implying that the extra L1 dynamic energy is about 3% of the L1 leakage energy, under these extreme assumptions. This assertion implies that if a DRI i-cache achieves sizable savings in leakage, the extra L1 dynamic energy will not outweigh the savings.

(Extra L2 dynamic energy)/ (L1 leakage energy)= [3.6 x (extra L2 accesses)]/ [('On' fraction) x 0.91 x (cycles)]
- [(3.95 / ('On" fraction)]x [(extra L2 accesses)/ (cycles)]
- [(3.95 / ("On" fraction)]x (extra L1 miss rate)
- 0.16 (if "on" fraction == 0.25, extra L1 miss rate == 1%)

Now we compare the extra L2 dynamic energy against the L1 leakage energy by computing their ratio. As, before, we simplify this ratio by approximating the number of cycles to be equal to the total number of L1 accesses, which allows us to express the ratio as a function of the ***absolute*** increase in the L1 miss rate (i.e., number of extra L1 misses divided by the total number of L1 accesses). If the "on" portion is as small as half the original size, and the absolute increase in L1 miss rate is as high as 1% (e.g., L1 miss rate increases from 5% to 6%), the ratio reduces to 0.16, implying that the extra L2 dynamic energy is about 16% of the L1 leakage energy, under these extreme assumptions. This assertion implies that if a DRI icache achieves sizable savings in leakage, the extra L2 dynamic energy will not outweigh the savings.

5.3 DRI I-Cache Energy Savings

In this section, we present the overall energy-savings achieved by a DRI i-cache. Because a DRI icache's energy dissipation depends on the miss-bound and size-bound, we show the best-case energy-savings achieved for each benchmark under various combinations of the miss-bound and size-bound. We determine the best case via simulation by empirically searching the combination space. We present the energy-delay product because it is a well-established metric used in low-power research. The energy-delay product ensures that both reduction in energy and accompanying degradation in performance are taken into consideration together, and not separately. But because the best-case energy-delay in some cases amounts to considerable performance degradation, which may be unacceptable in certain domains, we include performance-constrained, best-case energy-delay product by limiting overall performance degradation to under 4%, compared to a conventional i-cache.

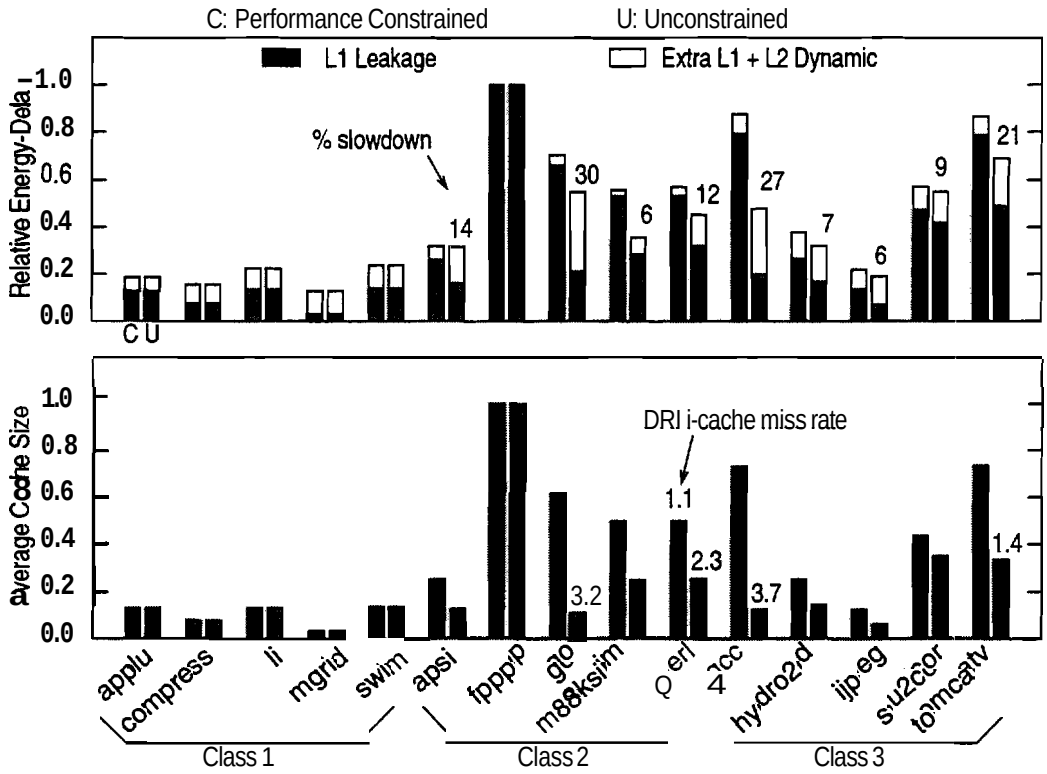

FIGURE 5: Performance-constrained and unconstrained best cases.

We compute the energy-delay product by multiplying the effective DRI i-cache leakage energy numbers from Section 5.2 with the execution time reported by SimpleScalar. Figure 5 (top graph) shows the effective energy-delay product normalized with respect to the conventional i-cache leakage energydelay product. We present the empirically-obtained best cases with performance degradation constrained to be within 4% relative to the conventional i-cache as well as unconstrained degradation. The stacked bars show the breakdown between the leakage and extra dynamic (L1 and L2) components. For the unconstrained case, we show the percentage increase in execution time relative to a conventional i-cache above the bars whenever performance degradation is worse than 4%.

Figure 5 (bottom graph) shows the DRI i-cache size averaged over the benchmark execution time, as a fraction of the conventional i-cache size. We show the miss rates under the unconstrained case above the bars whenever the miss rates are higher than 1%. Note that for the performance-constrained case, the DRI i-cache miss rates are all below 1%, except for *perl* at 1.1%. The conventional i-cache miss rate is less than 1% for all the benchmarks (highest being 0.7% for ***perl),*** indicating that a 64K-cache captures most of the working set of these benchmarks.

From the top graph, we see that a DRI i-cache achieves large reductions in the energy-delay product as performance degradation is constrained to be within 4%, demonstrating the effectiveness of our adaptive resizing scheme. In fact, out of the 15 benchmarks, 9 stay within 2% degradation. The reduction ranges from as much as 80% for ***applu, compress, ijpeg***, and ***mgrid,*** to 60% for ***apsi, hydro2d***, ***li***, and ***swim,*** 40% for ***m88ksim***, ***perl,*** and ***su2cor***, and 10% for ***gcc***, ***go,*** and ***tomcatv. Fpppp*** is the only benchmark with no reduction. The dynamic (extra L1+L2) component of the energy-delay product is small for all the benchmarks, implying that the extra L2 accesses are few enough not to increase the dynamic component significantly, as forecasted in Section 2.3 and Section 5.3.

For the unconstrained case, the best-case is identical to the performance-constrained case for many benchmarks. However, a few benchmarks ***(gcc***, ***go, m88ksim***, and ***tomcatv)*** have significantly lower energy-delay.

	Class 1					Class 2					Class 3				
	applu	*compress*	*li*	*mgrid*	*swim*	*apsi*	*fpppp*	*go*	*m88ksim*	*perl*	*gcc*	*hydro2d*	*ijpeg*	*su2cor*	*tomcatv*
conventional miss rate	6x 10^{-6}	1x 10^{-5}	3x 10^{-6}	8x 10^{-6}	2x 10^{-6}	2x 10^{-4}	5x 10^{-4}	9x 10^{-4}	5x 10^{-6}	7x 10^{-3}	5x 10^{-3}	2x 10^{-3}	2x 10^{-4}	4x 10^{-6}	6x 10^{-4}
miss rate (c)	8x 10^{-5}	3x 10^{-4}	3x 10^{-3}	7x 10^{-5}	2x 10^{-3}	2x 10^{-3}	5x 10^{-4}	4x 10^{-3}	8x 10^{-4}	1x 10^{-2}	9x 10^{-3}	2x 10^{-3}	2x 10^{-3}	2x 10^{-3}	3x 10^{-3}
miss-bound (c)	1x 10^{-3}	6x 10^{-4}	1x 10^{-2}	4x 10^{-3}	5x 10^{-3}	4x 10^{-2}	0	6x 10^{-3}	2x 10^{-3}	2x 10^{-2}	5x 10^{-3}	9x 10^{-3}	1x 10^{-2}	1x 10^{-3}	1x 10^{-3}
size-bound in KB (c)	8	4	8	2	4	16	4	32	32	32	8	2	8	4	8
miss-bound (u)	3x 10^{-3}	6x 10^{-4}	1x 10^{-2}	4x 10^{-3}	5x 10^{-3}	5x 10^{-2}	0	4x 10^{-1}	7x 10^{-3}	5x 10^{-2}	1x 10^{-1}	2x 10^{-2}	1x 10^{-2}	3x 10^{-3}	1x 10^{-2}
size-bound in KB (u)	8	4	8	2	4	8	64	8	16	16	64	8	2	4	8

Table 6: Miss-bound and size-bound for performance-constrained(c) and unconstrained(u) cases.

For all these benchmarks, performance of the unconstrained best-case is considerably worse than that of the conventional i-cache (e.g., *gcc* by 26%, go by 30%, tomcatv by 21%), indicating that the lower energy-delay product is achieved at the cost of performance. This observation is borne out by the fact that the dynamic component for these benchmarks is significantly larger, implying that the unconstrained case reduces leakage by downsizing the cache incurring numerous extra L1 misses.

From the bottom graph, we see that the average DRI i-cache size is significantly smaller than the conventional i-cache in the performance-constrained case, confirming our hypothesis that i-cache requirements vary both within and across applications. The average cache size reduction ranges from as much as 80% for applu, compress, ijpeg, li, and *mgrid*, to 60% for *m88ksim*, *perl*, and *su2cor*, and 20% for *gcc*, go, and tomcatv. For all the benchmarks, the miss rates continue to be less than 1%, staying close to those of the conventional i-cache, demonstrating the success of our adaptive scheme in downsizing to the required size while keeping the extra L1 misses in check. The only cases where the DRI i-cache miss rates are much higher than those of the conventional i-cache are under the unconstrained case for *gcc*, go, perl, and tomcatv, which downsize the cache to the extent of incurring numerous extra L1 misses.

In Table 6, we list the combinations of the miss-bound and size-bound corresponding to the best-case under performance-constrained and unconstrained cases for each benchmark. Because each benchmark's level of sensitivity to the miss-bound and size-bound is different, requiring different miss-bounds and size-bounds to determine the best-case. Even for the performance-constrained case (and more so for the constrained case), the miss-bounds ("miss-bound(c)" row) are one or two orders of magnitude higher than the conventional miss rates ("reference miss rate" row), encouraging miss rates higher than the conventional i-cache.

DRI i-cache's simple adaptive scheme enables the cache to downsize without causing the miss rate ("miss-rate(c)" row) to exceed the miss-bound, except for the relatively small excesses in the case of *gcc* and *su2cor*. Because the absolute differences between the conventional and DRI i-cache miss rates are still small in magnitude, the actual number of extra L1 misses is small and the performance loss is minimal. The largest miss-rate difference is 0.004 for *gcc* and from the calculations done in Section 5.3, we know that the this miss rate difference contributes only a small amount of extra L2 dynamic energy. This observation makes it profitable to trade off extra L2 dynamic energy for L1 leakage energy savings.

To understand the average i-cache size requirements better, we categorize the benchmarks into three classes. Benchmarks in the first class primarily require a small i-cache throughout their execution. Mostly, they execute tight loops allowing a DRI i-cache to stay at the size-bound, causing the unconstrained and the performance-constrained best-cases to match. Applu, compress, li, *mgrid* and swim fall in this class, and they almost always stay at the minimum size allowed by the size-bound. The dynamic component is a large

fraction of the DRI i-cache energy in these benchmarks because much of the L1 leakage energy is eliminated through size reduction and a large number of resizing tag bits are used to allow a small size-bound.

The second class consists of the benchmarks that primarily require a large i-cache throughout their execution and do not benefit much from downsizing. *Apsi*, *fpppp*, *go*, *m88ksim* and *perl* fall under this class, and *fpppp* is an extreme example of this class. If these benchmarks are encouraged to downsize via high miss-bounds, they incur a large number of extra L1 misses, resulting in significant performance loss. Consequently, the performance-constrainedcase uses a small number of resizing tag bits, forcing the size-bound to be reasonably large. *Fpppp* requires the full-sized i-cache, so reducing the size dramatically increases the miss rate, canceling out any leakage energy savings. Therefore, fpppp is disallowed from downsizing the cache by having a 64K size-bound. The rest of the applications are not as extreme as *fpppp*. At the performance-constrained best-case, the dynamic energy overhead is much less than the leakage energy savings, and allowing more downsizing is still beneficial. However, due to their large i-cache size requirements, the unconstrained best-case, obtained by downsizing beyond the performance-constrained best-case, falls outside the acceptable performance range.

The last class of applications exhibit distinct phases with diverse i-cache size requirements. *Gcc*, *hydro2d*, *ijpeg*, *su2cor* and *tomcatv* belong to this class of applications. A DRI i-cache's effectiveness to adapt to the required i-cache size is dependent on its ability to detect the program phase transitions and resize appropriately. *Hydro2d* and *ijpeg* fall into the group that have relatively clear phase transitions. After the initialization phase requiring the full size of i-cache, *hydro2d* consists mainly of small loops requiring only 2K of i-cache. *Ijpeg* follows this pattern. Therefore, a DRI i-cache adapts to the phases of *hydro2d* and *ijpeg* well, achieving small average sizes with little performance loss. The phase transitions in *gcc*, *su2cor* and *tomcatv* are not as clearly defined, resulting in a DRI i-cache not adapting as well as it did for *hydro2d* or *ijpeg*. Consequently, these benchmarks' best-case average sizes under both the performance-<:onstrainedand unconstrained case are relatively large.

5.4 Effect of Miss-Bound and Size-Bound

In this section, we present the effect of varying the miss-bound and size-bound on the energy-delay product. The miss-bound and size-bound are key parameters which determine the L2 and extra L1 dynamic energy, respectively.

5.4.1 Effect of Miss-Bound

The miss-bound controls the number of extra L1 misses caused by downsizing and directly affects the extra L2 dynamic energy dissipated to service the extra L1 misses. A higher miss-bound encourages a DRI i-cache to downsize despite a larger number of L1 misses at the current size.

Figure 6 shows the result of varying the miss-bound around the value corresponding to the performance-constrained best-case in Table 6. The size-bound is fixed at the same value as the performance-constrained best-case from Table 6. The center bar corresponds to the best-case miss-bound; the left and right bars correspond to half and twice of the center bar's miss-bound, respectively. Varying the miss-bounds at half and twice the best-case miss-bound keeps the resulting miss-bounds at reasonable values. The top graph shows the effective energy-delay product normalized to the conventional i-cache leakage energy-delay, and also the percentage performance-degradation for the cases which are 4% worse than the conventional i-cache. The bottom graph shows average cache size as a fraction of the conventional i-cache size and also the miss rate for the cases which are above 1%.

The energy-delay graph shows that despite varying the miss-bound over a factor of four range (i.e., from 0.5x to 2x), most of the benchmarks' energy-delay product does not change significantly. Even when the miss-bound is doubled (right bars), the L1 miss rates stay within 1% and the L2 dynamic energy-delay

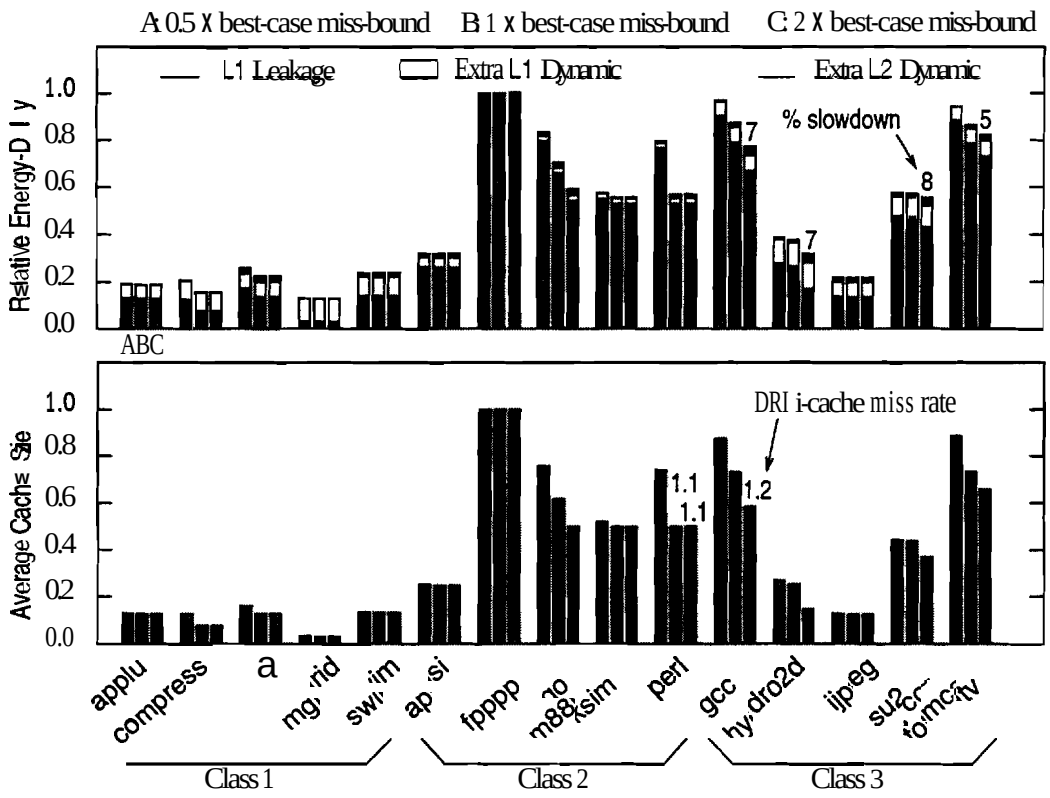

FIGURE 6: Effect of varying the miss-bound.

component does not increase much for most of the benchmarks. This behavior indicates that our adaptive scheme is fairly robust with respect to a reasonable range of miss-bounds. The exceptions are *gcc*, go, *perl*, and *tomcatv*, which need large i-caches but allow more downsizing under higher miss-bounds, as can be seen from the average size graph, because the application phase transitions are not readily identified by a DRI i-cache. These benchmarks achieve average sizes smaller than those of the best-cases, but incur more than 4%, albeit less than 10%, performance degradation, compared to the conventional i-cache.

5.4.2 Size-Bound

The number of resizing tag bits determines the size-bound to which the cache may be downsized, indirectly controlling the number of extra L1 misses and the extra L2 dynamic energy. The resizing tag bits directly incur extra L1 dynamic energy because the bits require charging and discharging additional bitlines (as discussed in Section 2.3). Each additional resizing tag bit decreases the size-bound by half, and is beneficial only if the leakage energy savings achieved by downsizing to the next half size is more than the extra L1 energy for the resizing tag bitlines and the extra L2 energy due to the extra L1 misses caused by such downsizing.

Figure 7 shows the effect of varying the size-bound to be double and half the value of the performance-constrained best-case in Table 6. The miss-bound is set at the value of the performance-constrained best-case in Table 6. The center bar for each benchmark except *fpppp*, corresponds to the best-case size-bound; the left and right bars correspond to double and half that size-bound, respectively. Fpppp's best-case size-bound is 64K, and therefore there is no left bar. The top graph shows the effective energydelay product normalized to the conventional i-cache leakage energy-delay and also the percentage slowdown for the cases which are 4% worse than the conventional i-cache. The bottom graph shows average cache size as a fraction of the conventional i-cache size and also the miss rate for the cases which are above 1%.

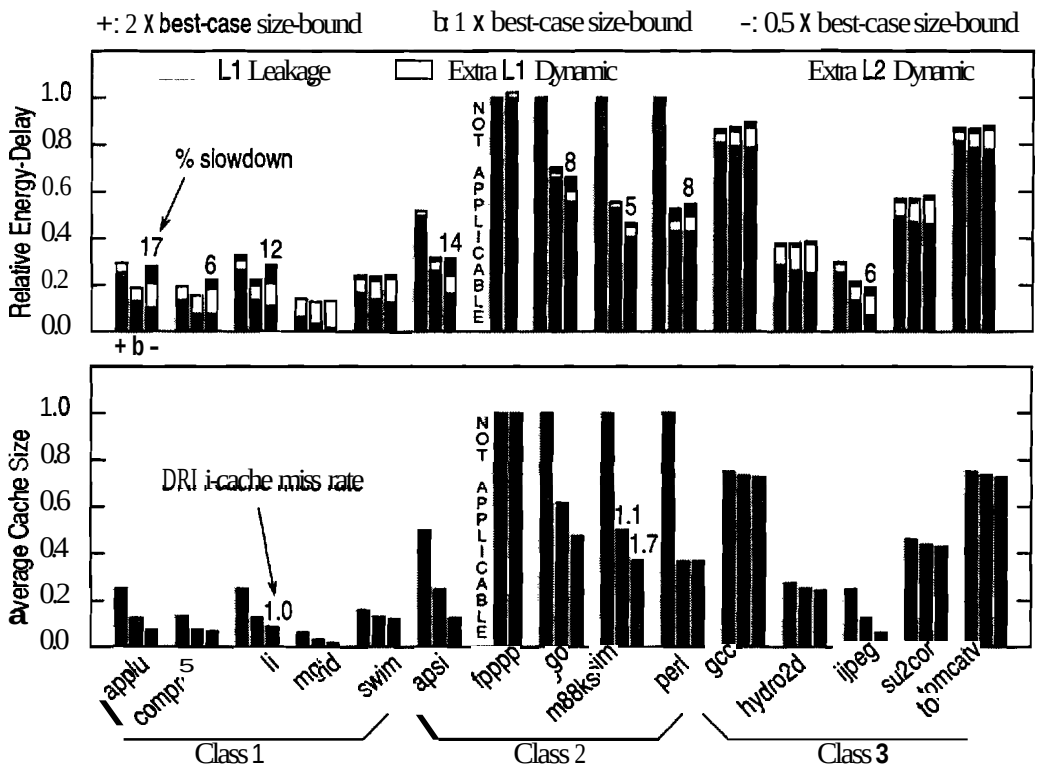

FIGURE 7: Effect of varying the size-bound.

For all benchmarks, a smaller size-bound results in larger reduction in the average cache size, but the effect on the energy-delay varies depending on the class to which the benchmark belongs. The first class of applications, applu, compress, li, *mgrid*, and swim, incur little performance degradation with the best-case size-bound because the benchmarks' i-cache requirements are small. Throughout the benchmarks' execution, a DRI i-cache stays at the minimum size allowed by the size-bound. Doubling the size-bound of the best-case (left bars) results in worse energy-delay than the best-case, corroborated by the fact that the average size is almost double that of the best-case. Halving the size-bound of the best-case (right bars), causes numerous extra L1 misses and increased extra L2 dynamic energy, which results in worse energy-delay.

Decreasing the size-bound for the second class, apsi, go, *m88ksim*, *perl*, which has relatively large i-cache requirements, encourages downsizing at the expense of performance. With a smaller size-bound, this class achieves lower energy-delay than the best-case by downsizing more, but incurs performance degradation beyond 4%. For the third class of applications, *gcc*, *hydro2d*, *ijpeg*, *su2cor*, and *tomcatv*, and *fpppp*, the extra L1 dynamic energy incurred by decreasing the size-bound beyond the best-case outstrips the leakage energy savings, resulting in higher energy-delay than the best-case. Using a 32K size-bound, Fpppp has worse energy-delay than a conventional i-cache, indicating that poor choice of parameters may result in a DRI i-cache having worse energy-delay than a conventional i-cache.

5.5 Effect of Conventional Cache Parameters

In this section, we investigate the impact of conventional cache parameters, size and associativity, on a DRI i-cache. Intuitively, higher associativity reduces conflict misses, making the miss rate less sensitive to cache size. Consequently, a DRI i-cache should be more effective in reducing the cache size without incurring many extra L1 misses and additional L2 dynamic energy. Because a DRI i-cache downsizes to the working set size of the application independent of the original size of the cache, increasing the size should result in larger energy savings.

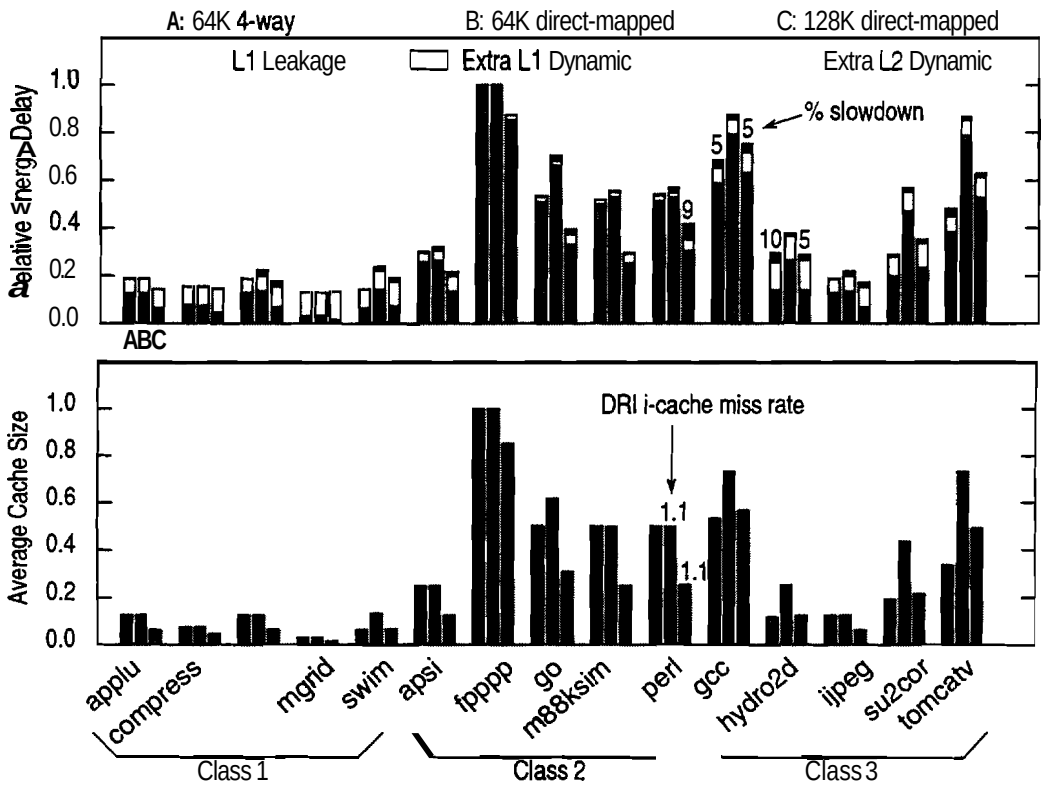

FIGURE 8: Effect of varying conventional cache parameters.

Figure 8 displays the results for a 64K, 4-way associative DRI i-cache, a 64K, direct-mapped DRI i-cache (as in Section 5.3), and a 128K, direct-mapped DRI i-cache, shown from left to right. The miss-bound is set to be the same as that of the performance-constrained best-case in Table 6. The size-bound is also as in the performance-constrained best-case in Table 6. The 128K direct-mapped case uses one more resizing tag bit so that the size-bound is the same as the 64K direct-mapped case. Energy-delay, average size, and performance degradation shown in the figures are all relative to a conventional i-cache of equivalent size and associativity. Thus, each bar is normalized with respect to a different conventional cache, and the left and center bars correspond to the same size (64K) but different associativities. The center and right bars correspond to the same associativity (direct-mapped) but different sizes. Note that the right-most bar and left-most bars are not directly comparable.

For ***applu, apsi, compress, fpppp, ijpeg, li,*** and ***mgrid,*** varying the associativity (left and center bars) does not impact the relative energy-delay product or the average cache size. The reason for this behavior is that the direct-mapped DRI i-cache miss rates are not high to start with, making added associativity insignificant. Consequently, the direct-mapped DRI i-cache achieves the same average size as the 4-way associative DRI i-cache, resulting in identical energy-delay products. For the rest of the benchmarks, ***gcc***, ***go, hydro2d***, ***su2cor***, ***swim*** and ***tomcatv***, the direct-mapped DRI i-cache miss rates range from 0.17% for ***su2cor*** to 0.92% for ***gcc***, giving the 4-way cache an opportunity to absorb some of the conflict misses. Thus, the 4-way associative DRI i-cache achieves smaller average size and lower energydelay for these benchmarks using the same miss-bound as the direct-mapped DRI i-cache. Using the same miss-bound for the 4-way associative DRI i-cache as the direct-mapped DRI i-cache encourages more extra misses in the 4-way associative DRI i-cache compared to a conventional 4-way associative cache. Consequently, for ***gcc***, ***hydro2d***, and ***tomcatv***, however, the smaller average size comes at the cost of performance degradation beyond 4%.

Increasing the size from 64K (center bars) to 128 K (right bars) while allowing the same size-bound (i.e., one extra resizing tag bit for the 128K cache) gives higher savings in energy-delay, because a larger fraction of the cache is turned "off'. In all cases, except for ***fpppp*** and ***gcc***, the 128K cache is downsized to the

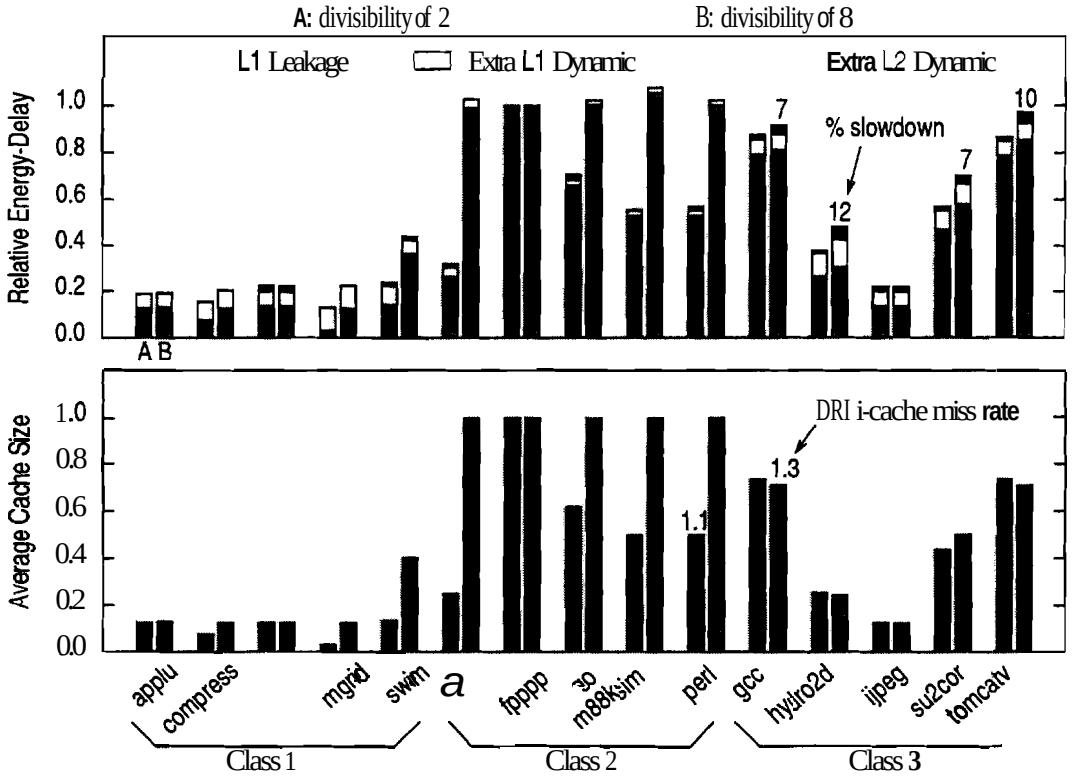

FIGURE 9: Effect of varying the divisibility.

same absolute magnitude as the 64K cache; when expressed as a fraction of the original size, the average size for the 128K DRI i-cache is half that for the 64K DRI i-cache. Fpppp and *gcc* are different because their working set sizes are larger than 64K in some (for *gcc*) or all (for *fpppp*) application phases, and so the 128K DRI i-cache does not downsize to 64K in those phases. Hence, the 128K DFU i-cache's average size is not half that of the 64K DRI i-cache. This shows that the DFU i-cache downsizes to the working set size of the application, regardless of the original cache size. For perl, *gcc*, and *hydro2d*, using the same miss-bound for the 128K cache as the 64K cache causes relatively more L1 misses in the 128K cache than the 64K cache, when compared with the respective equivalent conventional caches. The additional misses result in higher extra L2 dynamic energy and performance degradation worse than 4%, as indicated.

5.6 Effect of Adaptivity Parameters

A DRI i-cache is an adaptive system using the parameters: miss-bound, size-bound, interval length, and divisibility, and we have already evaluated the impact of the miss-bound and size-bound in the previous sections. Now, we look at the effect of interval length and divisibility on DRI i-cache behavior.

Ideally, we want the monitoring interval length to correspond to program phases, allowing the cache to resize before entering a new phase. Because we do not know the length of the phases, we approximate using a fixed monitoring interval length. Our experiments show that in almost all cases, a DRI i-cache is reasonably robust to interval length. More precisely, varying the interval length from 250K to 4M i-cache accesses, the energy-delay product varies by less than 1% except for go which shows a 5% difference.

Divisibility is used to control the rate of resizing by setting the factor by which the cache size changes per resizing. Large divisibility is favorable for both performance and energy consumption when the cache rapidly switches between large and small working sets. In that case, large divisibility reduces the time spent in the intermediate cache sizes. However, larger divisibility makes it difficult for the cache to adapt to the optimal size because larger divisibility makes the granularity of the resizing process coarser. Figure 9

shows that divisibility of 8 always exhibits worse energydelay products than those of divisibility of 2. For all the benchmarks, the coarser granularity prevents the cache from downsizing to the next one-eighth size either because of exceeding the miss-bound or because of exhausting the resizing tag bits.

6 Related Work

There are a number of previous studies focusing on reducing the ***dynamic*** power and energy dissipation in processors and cache memories. Manne, et al., [18] propose gating the processor pipeline stages to reduce the processor's dynamic energy dissipation. Brooks and Martonosi [8] propose exploiting; narrow-width operands to reduce the processor's dynamic power. Toburen, et al., [19] reduce processor dynamic power via instruction scheduling.

To reduce dynamic power of caches, Albonesi [2] proposes Selective Cache Ways, a mechanism to vary cache associativity dynamically by activating and deactivating cache banks in set-associative caches. There are several key differences between this technique and ours. This technique does not work for direct-mapped caches which are widely used due to their access speed advantages. Because this technique can vary the associativity and size only together, and not separately, this technique results in a greater number of extra misses than our technique, which varies only the size. While our adaptive scheme gives DRI i-caches tight control over the number of extra misses, this technique varies associativity which is a coarse-grained resizing approach that increases both capacity and conflict misses in the cache.

Kin, et al., [16] propose the Filter Cache, small (e.g., 256-byte) direct-mapped IO caches, that filter accesses to L1 I- and D-Caches and reduce power. Filter caches trade-off potentially significant degrees of performance loss for power savings. Similarly, Bellas, et al., propose the Loop-Cache and use the compiler [4,5] or hardware [4,5] to detect and place small loops in it to reduce the access frequency to the L1 i-cache. None of the previous studies have focused on reducing the leakage power in deep-submicron caches.

There are a number of previous studies that have focused on circuit-level only techniques to reduce leakage power. Several circuit techniques such as multiple transistor threshold voltages [20,17,25,30] and transistor stacking [32,30] have been used to reduce leakage power dissipation while maintaining high performance. More recently, dynamic transistor threshold control to achieve high performance when the transistor is "on" and low leakage current when the transistor is "off' [3,29] has been used for both bulk Silicon and Silicon on Insulator (SOI) technology. Multiple supply voltages with multiple transistor threshold voltages can also be used to achieve both dynamic and leakage power reduction [12,27]. However, circuit-level techniques that apply leakage reduction ignore application/architectural behavior and circuit utilization. Instead, we propose an integrated architectural and circuit-level approach to maximize opportunity for leakage reduction with minimal impact on performance.

7 Conclusions

This paper explored an integrated architectural and circuit-level approach to reduce leakage energy dissipation in deep-submicron cache memories while maintaining high performance. The key observation in this paper is that the demand on cache memory capacity varies both within and across applications. Modem caches, however, are designed to meet the worst-case application demand, resulting in poor utilization of on-chip caches, and consequently a energy inefficiency. We introduced a novel cache called the Dynamically Resizable i-cache (DRI i-cache) that dynamically reacts to application demand and adapts to the required cache size during an application's execution. At the circuit-level, the DRI i-cache employs a novel mechanism, called gated-V_{dd}, which effectively turns off the supply voltage to the SRAM cells in the DRI i-cache's unused sections, virtually eliminating leakage in these sections. Our adaptive scheme gives DRI i-caches tight control over the number of extra misses caused by resizing, enabling the DRI i-cache to contain both performance degradation and extra energy dissipation due to increased number of accesses to lower cache levels.

We evaluated and presented detailed simulation results from running the SPEC95 applications on a SimpleScalar model of a DRI i-cache used in an out-of-order engine. The results indicated that our adaptive scheme closely captures the application's working set size accurately, enabling a 64K DRI i-cache to reduce, on average, the size by 62% with performance degradation constrained within 4%, and by 78% with higher performance degradation. Compared to a conventional i-cache, a DRI i-cache reduces the leakage energy-delay product by 62% with performance degradation constrained within 4%, and by 67% with higher performance degradation. Because higher associativities encourage more downsizing, and larger sizes imply larger relative size reduction, DRI i-caches achieve even better energydelay products with higher associativity and larger size. We also showed that our adaptive scheme is robust in that it is not sensitive to many of the adaptivity parameters and performs predictably without drastic reactions to varying the rest of the adaptivity parameters.

References

[1] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and T.N. Vijaykumar. Gated-Vdd: A Circuit Mechanism for Reducing Leakage in Cache Memories. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, July 2000.

[2] D. H. Alboensi. Selective cache ways: On-demand cache resource allocation. In *32nd Annual IEEUACM International Symposium on Microarchitecture (MICRO 32)*, Nov. 1999.

[3] F. Assaderaghi, D. Sinitsky, S. Park, J. Bokor, P. K.Ko, and C. Hu. A dynamic threshold voltage MOSFET(DTMOS) for ultra-low voltage operation. *IEDM Digest*, page 809, 1994.

[4] N. Bellas, I. Hajj, and C. Polychronopoulos. Architectural and compiler support for energy reduction in the memory hierarchy of high performance microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, August 1998.

[5] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic management techniques to reduce energy in high-performance processors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, August 1999.

[6] S. Borkar. Private communication.

[7] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.

[8] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. Jan. 1999.

[9] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, July 2000.

[10] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[11] B. Davari, R. Dennard, and G. Shahidi. CMOS scaling for high performance and low power- the next ten years. *Proceedings of the IEEE*, 83(4):595, June 1995.

[12] M. Hamada et al. A top-down low power design technique using cluster volatge scaling with variable supply voltage scheme. In *Proceedings of the 1998 Custom Integrated Circuits Conference*, pages 495–498, 1998.

[13] F. Hamzaoglu, Y. Ye, A. Keshavarzi, K. Zhang, S. Narendra, S. Borkar, M. Stan, and V. De. Dual-vt sram cells with full-swing single-ended bit line sensing for high-performance on-chip cache in 0.13um technology generation. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, July 2000.

[14] C. Hu. *Low Power Design Methodologies*, chapter Device and technology impact on low power electronics, pages 21–35. Kluwer Publishing, 1996.

[15] M. B. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 1997.

[16] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *30th Annual IEEUACM International Symposium on Microarchitecture (MICRO 30)*, pages 184–193, December 1997.

[17] U. Ko, A. Pa, A. Hill, and P. Srivastava. Hybrid dual-threshold design techniques for high performance processors with low power features. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 307–311, 1998.

[18] S. Manne, A. Klauser, and D. Grunwald. Pipline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.

[19] T. M. C. Mark C. Toburen and M. Reilly. Instruction scheduling for low power disspiation in high performance microprocessors. In *Proceedings of the Power Driven Microarchitecture Workshop*.

[20] S. Mutoh et al. I-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE Journal of Solid-State Circuits*, 30(8):847–854, 1995.

[21] J.-K. Peir, Y. Lee, and W. W. Hsu. Capturing dynamic memory reference behavior with adaptive cache topology. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, pages 240–250, Oct. 1998.

[22] J. M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.

[23] Semiconductor Industry Association. The International Technology Roadmap for Semiconductors (ITRS). http://

www.semichips.org, 1999.

[24] D. Singh and V. Tiwari. Power challenges in the internet world. Cool Chips Tutorial in conjunction with the 32nd Annual International Symposium on Microarchitecture, November 1999.

[25] S. Sirichotiyakul et al. Standby power minimization through simultaneous threshold voltage selection and circuit sizing. In *Proceedings of the 36th Design Automation Conference,* 1999.

[26] L. Su et al. A high performance sub-0.25um cmos technology with multiple thresholds and copper interconnects. In *IEEE Symposium on VLSI Technology,* 1998.

[27] K. Usami and M. Horowitz. Design methodology of ultra low-power mpeg4 codec core ecploiting voltage scaling techniques. In *Proceedings of the 35th Design Automation Conference,* pages 483–488, 1998.

[28] L. Wei, Z. Chen, M. C. Johnson, K. Roy, and V. De. Design and optimization of low voltage high performance dual threshold CMOS circuits. In *Proceedings of the 35th Design Automation Conference,* pages 489–494, 1998.

[29] L. Wei, Z. Chen, and K. Roy. Double gate dynamic threshold voltages (DGDT) SOI MOSFETs for low power high performance designs. In *ZEEE International SOI Conference,* 1997.

[30] L. Wei and K. Roy. Design and optimization for low-leakage with multiple threshold CMOS. In *ZEEE Workshop on Power and Timing Modeling.* pages 3–7, Oct. 1998.

[31] S. J. E. Wilson and N. P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report 93/5, Digital Equipment Corporation, Western Research Laboratory, July 1994.

[32] Y. Ye, S. Borkar, and V. De. A new technique for standby leakage reduction in high performance circuits. In *IEEE Symposium on VLSI Circuits,* pages 40–41, 1998.