Appears in **IEEE TVLSI special issue on low-power design**, February 2001.

An Energy-Efficient High-Performance Deep-Submicron Instruction Cache

Michael D. Powell[ϒ], Se-Hyun Yang[β1], Babak Falsafi[β1], Kaushik Roy[ϒ], and T. N. Vijaykumar[ϒ]

[ϒ]School of Electrical and Computer Engineering
Purdue University
{mdpowell,kaushik,vijay}@ecn.purdue.edu

[β]Electrical and Computer Engineering Department
Carnegie Mellon University
{syang,babak}@ece.cmu.edu

http://www.ece.purdue.edu/~icalp

Abstract

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 subthreshold leakage current even when the transistor is not switching. Estimates suggest a five-fold increase in leakage energy in every future generation. In modern microarchitectures, much of the leakage energy is dissipated in large on-chip cache memory structures with high transistor densities. While cache utilization varies both within and across applications, modern cache designs are fixed in size resulting in transistor leakage inefficiencies.

This paper explores an integrated architectural and circuit-level approach to reducing leakage energy in instruction caches (i-caches). At the architecture level, we propose the *Dynamically ResIzable* i-cache (DRI i-cache), a novel i-cache design that dynamically resizes and adapts to an application's required size. At the circuit-level, we use gated-V_{dd}, a novel mechanism that effectively turns off the supply voltage to, and eliminates leakage in, the SRAM cells in a DRI i-cache's unused sections. Architectural and circuit-level simulation results indicate that a DRI i-cache successfully and robustly exploits the cache size variability both within and across applications. Compared to a conventional i-cache using an aggressively-scaled threshold voltage a 64K DRI i-cache reduces on average both the leakage energy-delay product and cache size by 62%, with less than 4% impact on execution time. Our results also indicate that a wide NMOS dual-V_t gated-V_{dd} transistor with a charge pump offers the best gating implementation and virtually eliminates leakage energy with minimal increase in an SRAM cell read time area as compared to an i-cache with an aggressively-scaled threshold voltage.

Keywords: Cache memories, adapative systems, computer architecture, energy management, leakage currents.

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 accompanied by an increase in chips' energy dissipation. Higher energy dissipation requires more expensive packaging and cooling technology, increases cost, and decreases reliability of products in all segments of computing market from portable systems to high-end servers [21]. Moreover, higher energy dissipation significantly reduces battery life and diminishes 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 a device switches. Chip designers have relied on scaling down the transistor supply voltage in subsequent generations to reduce this 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 [19]. The International Technology Roadmap for Semiconductors [20] 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 subthreshold leakage current even when the transistor is not switching [3,28,24,16,22,11,6]. Borkar [3] 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 caches and data caches, translation lookaside buffers, and prediction tables. For instance, 30% of Alpha 21264 and 60% of StrongARM are devoted to cache and memory structures [14]. 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 30% of L1 cache energy and as much as 80% of L2 cache energy [7]. Unfortunately, current proposals for energy-efficient cache architectures [13,2,1] 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 [26,22,16] or multi-supply [9,23] voltage designs, dynamic threshold [25] or dynamic supply [4] voltage designs, transistor stacking [28], and

[1] This work was performed when Se-Hyun Yang and Babak Falsafi were at the School of Electrical and Computer Engineering at Purdue University.

cooling [3]). These techniques, however, typically impact circuit performance and are only applicable to circuit sections that are not performance-critical [10]. Second, unlike embedded processor designs [15,8], techniques relying only on multiple threshold voltages may not be as effective in high-performance microprocessor designs, where the range of offered supply voltages is limited due to gate-oxide wear-out and reliability considerations [10]. Third, techniques such as dynamic supply- and threshold-voltage designs may require a sophisticated fabrication process and increase cost. 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 the off-chip access frequency and improve performance. Modern cache hierarchies are designed to satisfy the demands of the most memory-intensive applications or application phases. The actual cache hierarchy utilization, however, varies widely both *within* and *across* applications. Recent studies on block frame utilization in caches [17], 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 reducing leakage energy dissipation in deep-submicron cache memories. We propose a novel instruction 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 cache's unused sections to eliminate leakage. At the architectural level, a DRI i-cache relies on simple techniques to exploit 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 DRI i-cache uses a mechanism we recently proposed, *gated-V_{dd}* [18], which reduces leakage by effectively turning off the supply voltage to the SRAM cells of the cache's unused block frames. Gated-V_{dd} may be implemented using NMOS or PMOS transistors, presenting a trade-off among area overhead, leakage reduction, and impact on performance. By curbing leakage, gated-V_{dd} enables high performance through aggressive threshold-voltage-scaling, which has been considered difficult due to inordinate increase in leakage.

We use cycle-accurate architectural simulation and circuit tools for energy estimation, and compare a DRI i-cache to a conventional i-cache using an aggressively-scaled threshold voltage to show that:

- 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 this variability and reduces the average size of a 64K cache by 62% with performance degradation constrained within 4%.
- Lowering the cell threshold voltage from 0.4V to 0.2V results in doubling the cell speed and two orders of magnitude increase in leakage. A wide NMOS dual-V_t gated-V_{dd} transistor with a charge pump offers the best gated-V_{dd} implementation and virtually eliminates leakage with only 8% cell read time and 5% area increase.
- A DRI i-cache effectively integrates architectural and the gated-V_{dd} circuit techniques to reduce leakage in an L1 i-cache. A DRI i-cache reduces the leakage energy-delay product by 62% with performance degradation within 4%, and by 67% with higher performance degradation.
- Our adaptive scheme gives a DRI i-cache tight control over the miss rate to keep it close to a preset value, enabling the DRI i-cache to contain both the performance degradation and the increase in lower cache levels' energy dissipation. Moreover, the scheme is robust and performs predictably without drastic reactions to varying 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. Finally, in Section 6 we conclude the paper.

2 DRI I-CACHE: REDUCING DEEP-SUBMICRON I-CACHE LEAKAGE

This paper describes the *Dynamically ResIzable 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} [18], to turn off the supply voltage to the cache's unused SRAM cells. In this section, we describe the anatomy of a DRI i-cache. In the next section, we 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 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 i-cache 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. 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 [1]. 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, chang-

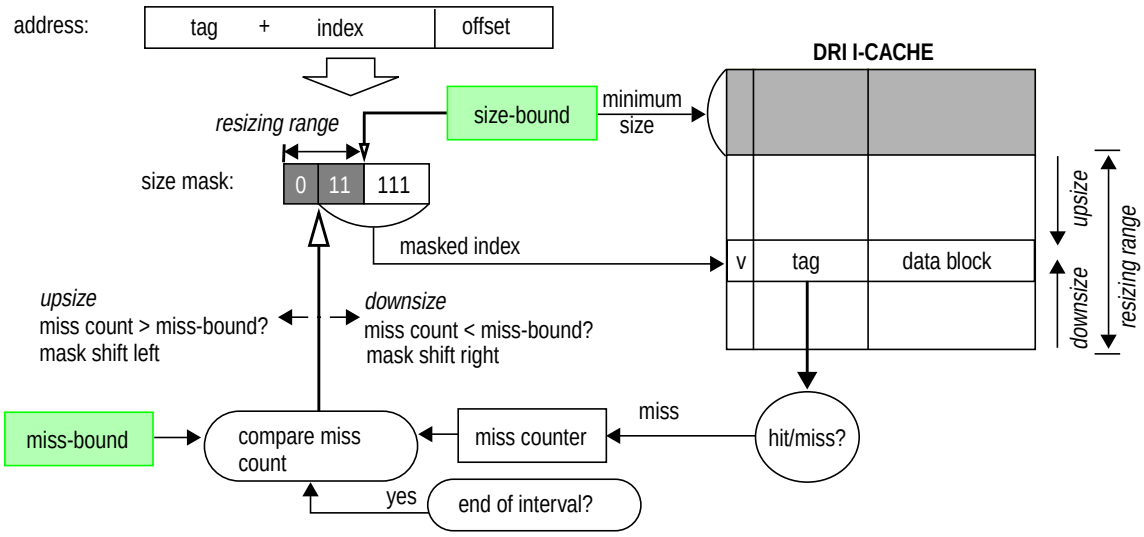

FIGURE 1: Anatomy of a DRI i-cache.

ing 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 data caches (d-caches), we focus on i-cache designs. Because of complications involving dirty cache blocks, studying d-cache designs is beyond the scope of this paper.

In the rest of this section, we first describe the basic DRI i-cache design and the adaptive mechanisms to detect application phase transitions and the required i-cache size. Next, we discuss the block lookup implications of a DRI i-cache. Finally, we present the impact of our design on energy dissipation and performance.

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. Figure 1 depicts the anatomy of a direct-mapped DRI i-cache (the same design applies to set-associative caches). To monitor cache performance, a DRI i-cache divides an application's execution time into fixed-length intervals, the *sense-intervals*, measured in the number of dynamic instructions (e.g., one million instructions). We use miss rate as the primary metric for monitoring cache performance. A miss counter counts the number of 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* (e.g., ten thousand misses). The factor by which the cache changes size is called the *divisibility*. A divisibility of two, for instance, changes the cache size upon upsizing/downsizing by a factor of two. To prevent the cache from thrashing and downsizing to prohibitively small sizes (e.g., 1K), the *size-bound* specifies the minimum size the i-cache can assume.

All the cache parameters can be set either dynamically or statically. Because this paper is a first step towards understanding a dynamically resizable cache design, 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 a DRI i-cache's performance. Intuitively, the sense-interval length allows selecting an interval length that best matches an application's phase transition times, and the divisibility determines the rate at which the i-cache is resized.

While the above parameters control the cache's aggressiveness in resizing, the adaptive mechanism may need throttling to prevent repeated resizing between two sizes if the desired size lies between the two sizes. We use a simple saturating counter to detect repeated resizing between two adjacent sizes. Upon detection, our mechanism prevents downsizing (while allowing upsizing) for a fixed number of successive intervals. This simple throttling mechanism works well in practice, at least for the benchmarks studied in this paper.

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.

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 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.

2.2 Implications on Cache Lookups

Using the resizing tag bits, we ensure that the cache functions correctly at every individual size. However, resizing 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 the 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. While a simple solution, flushing the cache or moving block frames to the appropriate places may incur prohibitively large 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).

Moreover, upsizing the cache may complicate lookup because blocks map to different sets in different cache sizes when upsizing the cache. Such a scenario creates two problems. A lookup for a block after upsizing fails to find it, and therefore fetches and places the block 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 the block in the cache. Unlike d-caches, however, in the common case a processor only 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. Fortunately, conventional systems often resort to flushing the i-cache in these cases because such scenarios are infrequent.

Compared to a conventional cache, the DRI i-cache has one extra gate delay in the index path due to the size mask (Figure 1), which may impact the cache lookup time. Because the size mask is modified at most only once every sense-interval, which is usually of the order of a million cycles, implementation of the extra gate level can be optimized to minimize delay. For instance, the size mask inputs to the extra gate level can be set up well ahead of the address, minimizing the index path delay. Furthermore, the extra gate level can also be folded into the address decode tree of the cache's tag and data arrays. Hence, in the remainder of the paper we assume that the extra gate delay does not significantly impact the cache lookup time.

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 resulting 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 [1]. We present energy calculations in Section 5.2.1 to show that for a DRI i-cache to cause significant increase in the L2 dynamic energy, the extra L1 misses have to be considerably large in number. In Section 5.3, we present experimental results that indicate that the extra L1 misses are usually small in number.

In addition to potentially increasing the L2 dynamic energy, a DRI i-cache may dissipate more dynamic energy due to the resizing tag bits, as compared to a conventional design. We present energy calculations in Section 5.2.1 and experimental results in Section 5.3 that indicate that the resizing tag bits have minimal impact on a DRI i-cache's energy.

Finally, the resizing circuitry may increase energy dissipation offsetting the gains from cache resizing. The counters required to implement resizing have a small number of bits compared to the cache, making their leakage negligible. Using the argument that the i^{th} bit in a counter switches once only every 2^i increments, we can show that the average number of bits switching on a counter increment is less than two. Thus the dynamic energy of the counters is also small. The dynamic energy dissipated to drive the resizing control lines can be neglected because resizing occurs infrequently (e.g., once every one million instructions).

2.3.1 Controlling Extra Misses

Because a DRI i-cache's miss rate impacts both energy and performance, the cache uses its key parameters to achieve tight control over its miss rate. We explain the factors that may cause a high miss rate and describe how the parameters control the miss rate.

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. Furthermore, the throttling mechanism helps reduce unnecessary switching, virtually eliminating frequent resizing between two adjacent sizes, in practice. Our results indicate that optimal interval lengths to match application phase transition times are long enough to 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 miss rate when the required cache size is slightly below a given size. The impact on the miss rate is highest at small cache sizes when the cache begins to thrash. A DRI i-caches uses the size-bound to guarantee a minimum size preventing the cache from thrashing.

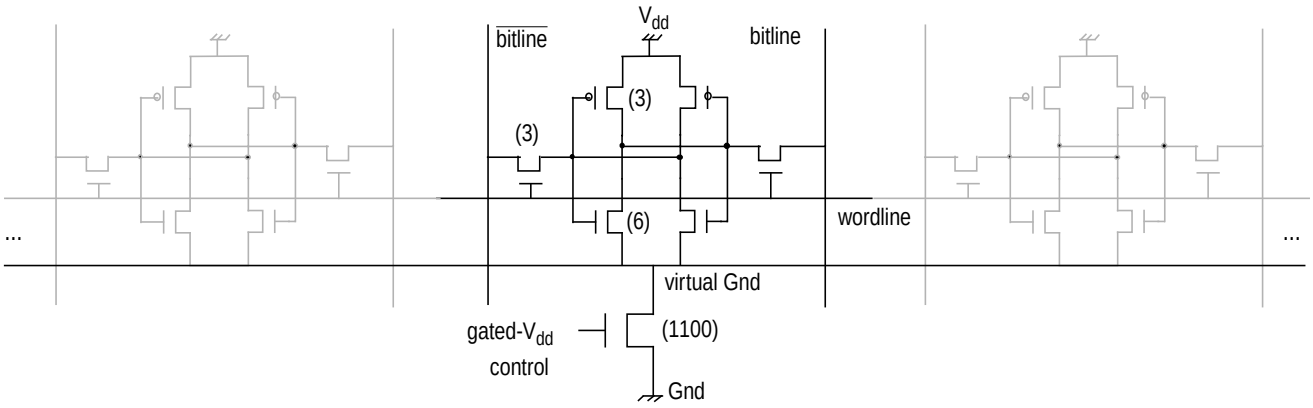

FIGURE 2: 6-T SRAM cells connected to a gated-V_{dd} transistor (typical transistor W/L ratios).

Miss-bound and size-bound control a DRI i-cache's aggressiveness in reducing the cache size and leakage 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 at the cost of high performance degradation. A conservative DRI i-cache configuration maintains a miss rate which is close to the miss rate of a conventional i-cache of the same base size, and bounds the downsizing to larger sizes to prevent thrashing and significantly increasing the miss rate. Such a configuration reduces leakage with minimal impact on execution time and dynamic energy.

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 larger divisibility favors applications with drastic changes in i-cache requirements, it makes size transitions more coarse reducing the opportunity to adapt closer to the required size. Similarly, while longer sense-intervals may span multiple application phases reducing opportunity for resizing, shorter intervals may result in higher overhead. Our results indicate that sense-interval and divisibility are less critical than miss-bound and size-bound to controlling extra misses (Section 5.3.3).

3 GATED-V_{DD}: CIRCUIT-LEVEL SUPPLY-VOLTAGE GATING

Current technology scaling trends [3] require aggressively scaling down the threshold voltage (V_t) to maintain transistor switching speeds. Unfortunately, there is a *subthreshold leakage* current through transistors that increases exponentially with decreasing threshold voltage, resulting in a significant amount of *leakage energy* dissipation at a low threshold voltage.

To prevent the leakage energy dissipation in a DRI i-cache from limiting aggressive threshold-voltage scaling, we use a circuit-level mechanism called *gated-V_{dd}* [18]. 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 to the ground 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.

Rather than gating the cells, many embedded designs [15] use circuit-only techniques [8] and primarily rely on a dual-threshold voltage (dual-V_t) process technology [24] to reduce leakage. Dual-V_t allows integrating transistors with two different threshold voltages. These designs use high V_t and V_{dd} for the cell transistors (which account for much of the leakage energy) and use low V_t and V_{dd} for the transistors in the rest of the cache (to maintain low read/write delay and low switching energy). However, the voltage spread between the high V_{dd} and low V_{dd} in such dual-V_t designs may be large. Unfortunately, unlike embedded designs, in high-performance designs the range of offered supply voltages is limited due to gate-oxide wear-out and stability considerations [10], reducing the effectiveness of dual-V_t alone in eliminating leakage. By providing an alternative solution, our integrated circuit/architecture approach to reducing leakage for high-performance designs [18] offers a key advantage over the dual-V_t approach.

The fundamental reason why gated-V_{dd} achieves significantly lower leakage is that two off transistors connected in series reduce the leakage current by orders of magnitude; this effect is due to the self reverse-biasing of stacked transistors, and is called the *stacking effect* [28]. The gated-V_{dd} 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.

Figure 2 depicts the anatomy of conventional 6-T SRAM cells 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 through the pass transistors. Each of the two inverters have a V_{dd} to Gnd leakage path through a pair of series-connected NMOS and PMOS transistors, one of which is turned off. 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. When the gated-V_{dd} transistor is off, it is in series with the off inverter transistors, producing the stacking effect. The 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 SRAM cells from one or more cache blocks to amortize the overhead of the extra transistor (Figure 2). 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 active mode, too large a gated-V_{dd} transistor may reduce the stacking effect, thereby diminishing the energy savings. Moreover, large transistors also increase the area of overhead due to gating. Figure 2 shows the width/length ratios for cell and gated-V_{dd} transistors typically used in this paper.

Gated-V_{dd} can 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 among area overhead, leakage reduction, and impact on performance [18]. Moreover, gated-V_{dd} can be coupled with dual-V_t to achieve even larger reductions in leakage. With dual-V_t, the SRAM cells use low-V_t transistors to maintain a high speed while the gated-V_{dd} transistors use high V_t to achieve additional leakage reduction. Because the gated-V_{dd} transistor already exploits the stacking effect, the gated-V_{dd} transistor needs to use only marginally higher V_t to achieve further leakage reduction. Hence, the dual-V_t required for gated-V_{dd} is not likely to run into the previously-mentioned supply voltage spread problems. In Section 5.1.2, we evaluate various gated-V_{dd} implementations and show that NMOS gated-V_{dd} transistors with dual-V_t achieves a good compromise among performance, energy, and area [18].

4 Methodology

We use SimpleScalar-2.0 [5] to simulate an L1 DRI i-cache in the context of an out-of-order microprocessor. Table 1 shows the base configuration for the simulated system. We simulate a 1Ghz processor. We run all of SPEC95 with the exception of two floating-point benchmarks and one integer benchmark (in the interest of reducing simulation turnaround time).

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 1: System configuration parameters.

To determine the energy usage of a DRI i-cache, we use geometry and layout information from CACTI [27]. 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 operations on single rows and columns. We use this information to determine energy dissipation for appropriate cache configurations.

We use a Mentor Graphics IC-Station layout of a single cache line to estimate area. Figure 3 shows an example layout of 64 SRAM cells on the left and an adjoining NMOS gated-V_{dd} transistor. To minimize the area overhead and optimize layout, we implemented the gated-V_{dd} transistor as rows of parallel transistors placed along the length of the SRAM cells where each row is as long as the height of the SRAM cells. We obtain the desired gated-V_{dd} transistor width by varying the number of rows of transistors used, and estimate the area overhead accordingly.

All simulations use an aggressively-scaled supply voltage of 1.0V. We estimate cell read time and energy dissipation using Hspice transient analysis. We ensure that the SRAM cells are all initialized to a stable state before measuring read time or active mode leakage energy. We compute active and standby mode energy dissipation after the cells reach steady state with the gated-V_{dd} transistor in the appropriate mode. We assume the read time to be the time to lower the bitline to 75% of V_{dd} after the wordline is asserted.

5 Results

In this section, we present experimental results on the energy and performance trade-off of a DRI i-cache as compared to a conventional i-cache. First, we present detailed circuit results corroborating the impact of technology scaling trends on an SRAM cell's performance and leakage, and evaluate various gated-V_{dd} implementations. Second, we present our energy calculations and discuss the leakage and dynamic energy trade-off of a DRI i-cache. Finally, we present energy savings achieved for the benchmarks, demonstrating a DRI i-cache's effectiveness in reducing average cache size and energy dissipation, and the impact of a DRI i-cache's parameters on energy and performance.

5.1 Circuit Results

Because the key motivation for lowering the threshold voltage is higher performance, in this section we first analyze the impact of threshold voltage on performance and leakage. Then, we present experimental results to show the trade-off among leakage reduction, overall energy savings, and cell performance for the various gated-V_{dd} implementations (as discussed in Section 3).

5.1.1 Impact of Lowering Threshold Voltage

Table 2 shows the impact of lowering the threshold voltage on relative cell read time and leakage energy using NMOS gated-V_{dd} transistors. The relative cell read times are computed with respect to the cell and gated-V_{dd} transistor combination, both using a V_t of 0.2V. The first three rows indicate that decreasing the cell threshold voltage improves cell read time by more than a factor of two at the cost

FIGURE 3: Layout of 64 SRAM cells connected to a single gated-V_{dd} NMOS transistor.

SRAM Cell V_t (V)	Gated-V_{dd} V_t (V)	Relative Read Time	Active Leakage Energy (aJ)	Standby Leakage Energy (aJ)
0.40	0.40	2.8	12	10
0.30	0.40	2.3	143	49
0.20	0.40	1.1	1700	50
0.40	0.20	2.6	12	11
0.30	0.20	2.1	143	76
0.20	0.20	1.0	1700	165

Table 2: Lowering transistor threshold voltages.

of increasing the active leakage energy by several orders of magnitude. The standby column shows the standby mode leakage energy using gated-V_{dd} to be orders of magnitude smaller than active energy. Comparing the first three rows with the last three indicates that decreasing the threshold voltage of the gated-V_{dd} transistors significantly increases standby leakage energy dissipation.

5.1.2 Impact of Various Gated-V_{dd} Implementations

Increasing the gated-V_{dd} transistor width improves SRAM cell read times but decreases energy savings while increasing area. Table 3 shows energy, area, and relative speed as the width of the gated-V_{dd} transistor is increased. In the first row, the gated-V_{dd} transistor width is set as described in Section 3 and increased in the second and third rows. The cell and the gated-V_{dd} transistors threshold voltage is 0.20V for these simulations. There is a clear trade-off in cell read time against area and standby energy, though the standby energy is low in all cases.

Table 4 depicts the four circuit-level gated-V_{dd} implementations we evaluate. The table depicts the percentage of leakage energy saved in the standby mode, the cell read times, and the area overhead of each technique relative to a standard low-V_t SRAM cell with no gated-V_{dd}. The techniques can be grouped into two categories: the first category (the first three rows) has lower performance and the second (the last three rows) has higher performance.

From the first two rows we see that in spite of decreasing the cell threshold voltage from 0.40V to 0.20V, gated-V_{dd} manages to reduce the standby mode energy. The second and third rows indicate the trade-off between energy and speed depending on the threshold voltage of the gated-V_{dd} transistor. The fifth row indicates a slightly

Area Increase (%) of NMOS Gated-V_{dd}	Relative Read Time	Active Leakage Energy (aJ)	Standby Leakage Energy (aJ)
2	1.00	1700	166
4	0.90	1710	245
8	0.85	1720	371

Table 3: Widening the gated-V_{dd} transistor.

faster read time for gated-V_{dd} because the PMOS gated-V_{dd} transistor creates a virtual V_{dd} for the SRAM cells slightly lower than the supply voltage. Therefore, we may use PMOS gated-V_{dd} transistors to sacrifice energy savings for better performance.

To mitigate the negative impact on SRAM cell speed due to an NMOS gated-V_{dd} transistor, we can use a wider transistor with a charge pump. To offset a wider transistor's increased leakage current, we further raise the gated-V_{dd} transistor's threshold voltage. The last row shows results for increasing the gated-V_{dd} transistor width by a factor of four and adding a charge pump that raises the active mode gate voltage to 1.35V. The resulting SRAM speed overhead is only around 8% compared to the low threshold voltage SRAM cells without gated-V_{dd}, while the relative reduction in standby mode energy is 97%.

5.2 Energy Calculations

A DRI i-cache decreases leakage energy by gating V_{dd} to cache sections in standby mode but increases both L1 dynamic energy due to the resizing tag bits and L2 dynamic energy due to extra L1 misses. We compute the energy savings using a DRI i-cache compared to a conventional i-cache using an aggressively-scaled threshold voltage. Therefore,

energy savings = conventional i-cache leakage energy – effective L1 DRI i-cache leakage energy

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

L1 leakage energy = active portion leakage energy + standby portion leakage energy

active portion leakage energy = active fraction × conventional i-cache leakage energy

standby portion leakage energy ≈ 0

Implementation Technique	Gated-V_{dd} V_t (V)	SRAM V_t (V)	Relative Read Time	Active Leakage Energy (nJ)	Standby Leakage Energy (nJ)	Energy Savings (%)	Area Increase (%)
no gated-V_{dd}, high-V_t	N/A	0.40	2.22	50	N/A	N/A	N/A
NMOS gated-V_{dd}, dual-V_t	0.40	0.20	1.30	1690	50	97	2
NMOS gated-V_{dd}, dual-V_t	0.50	0.20	1.35	1740	49	97	2
no gated-V_{dd}, low-V_t	N/A	0.20	1.00	1740	N/A	N/A	N/A
PMOS gated-V_{dd}, low-V_t	0.20	0.20	1.00	1740	235	86	0
NMOS gated-V_{dd}, dual-V_t, wide, charge pump	0.40	0.20	1.08	1740	53	97	5

Table 4: Energy, speed, and area of various gated-V_{dd} implementations.

extra L1 dynamic energy = resizing bits ×
dynamic energy of 1 bitline per L1 access × L1 accesses

extra L2 dynamic energy = dynamic energy per L2 access ×
extra L2 accesses

The effective L1 leakage energy is the leakage energy dissipated by the DRI i-cache during the course of the application execution. This energy consists of three components. The first component, the L1 leakage energy, is the leakage energy dissipated in the active and standby portions of the DRI i-cache. We compute the active portion's leakage energy as the leakage energy dissipated by a conventional i-cache in one cycle times a DRI i-cache active portion size (as a fraction of the total size) times the number of cycles. We obtain the average active portion size and the number of cycles from Simplescalar simulations. Using the low-V_t active cell leakage energy numbers in Table 4, we compute the leakage energy for a conventional i-cache per cycle to be 0.91 nJ. Because the standby mode energy is a factor of 30 smaller than the active mode energy in Table 4, we approximate the standby mode term as zero. Therefore,

L1 leakage energy = active fraction × 0.91 × cycles

The second component is the extra L1 dynamic energy dissipated due to the 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. Using CACTI's Spice files, we estimate the dynamic energy per resizing bitline to be 0.0022 nJ. Therefore,

extra L1 dynamic energy = resizing bits × 0.0022 × L1 accesses

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. We use the calculations for cache access energy in [12] and estimate the dynamic energy per L2 access to be 3.6 nJ. Therefore,

extra L2 dynamic energy = 3.6 × extra L2 accesses

Using these expressions for L1 leakage energy, extra L1 dynamic energy, and extra L2 dynamic energy, we compute the effective L1 leakage energy and the overall energy savings of a DRI i-cache.

5.2.1 Leakage and Dynamic Energy Trade-off

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 the extra (L1+L2) dynamic energy, as forecasted in 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 × 0.0022) / (active fraction × 0.91) ≈
0.024 (if resizing bits = 5 and active fraction = 0.50)

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), 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 active 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 × extra L2 accesses) / (active fraction × 0.91 × cycles) ≈
(3.95 / active fraction) × extra L1 miss rate ≈
0.08 (if active fraction = 0.50 and extra L1 miss rate = 0.01)

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 active 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 0.08, implying that the extra L2 dynamic energy is about 8% 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 L2 dynamic energy will not outweigh the savings.

5.3 Overall Energy Savings and Performance Results

In this section, we present the overall energy savings achieved by a DRI i-cache. Unless stated otherwise, all the measurements in this section use a sense-interval of one million instructions and a divisibility of two. To prevent repeated resizing between two adjacent sizes (Section 2.1), we use a 3-bit saturating counter to trigger throttling and prevent downsizing for a period of ten sense-intervals.

Because a DRI i-cache's energy dissipation mainly depends on the miss-bound and size-bound, we show the best-case energy savings achieved under various combinations of these parameters. We determine the best case via simulation by empirically searching the combination space. Each benchmark's level of sensitivity to the miss-bound and size-bound is different, requiring different values to determine the best-case energy-delay. Most benchmarks, however, exhibit low miss rates in the conventional i-cache, and therefore tolerate miss-bounds that are one to two orders of magnitude higher than the conventional i-cache miss rates.

We present the energy-delay product because it ensures that both reduction in energy and the accompanying degradation in performance are taken into consideration together, and not separately. We present results on two design points. Our "performance-constrained" measurements focus on a DRI i-cache's ability to save energy with minimal impact on performance. Therefore, these measurements search for the best-case energy-delay while limiting the performance degradation to under 4% as compared to a conventional i-cache using an aggressively-scaled threshold voltage. The "performance-unconstrained" measurements simply search for the best-case energy-delay without limiting the performance degradation. We include performance-unconstrained measurements to show the best possible energy-delay, although the performance-unconstrained case sometimes amounts to prohibitively high performance degradation. We compute the energy-delay product by multiplying the effective

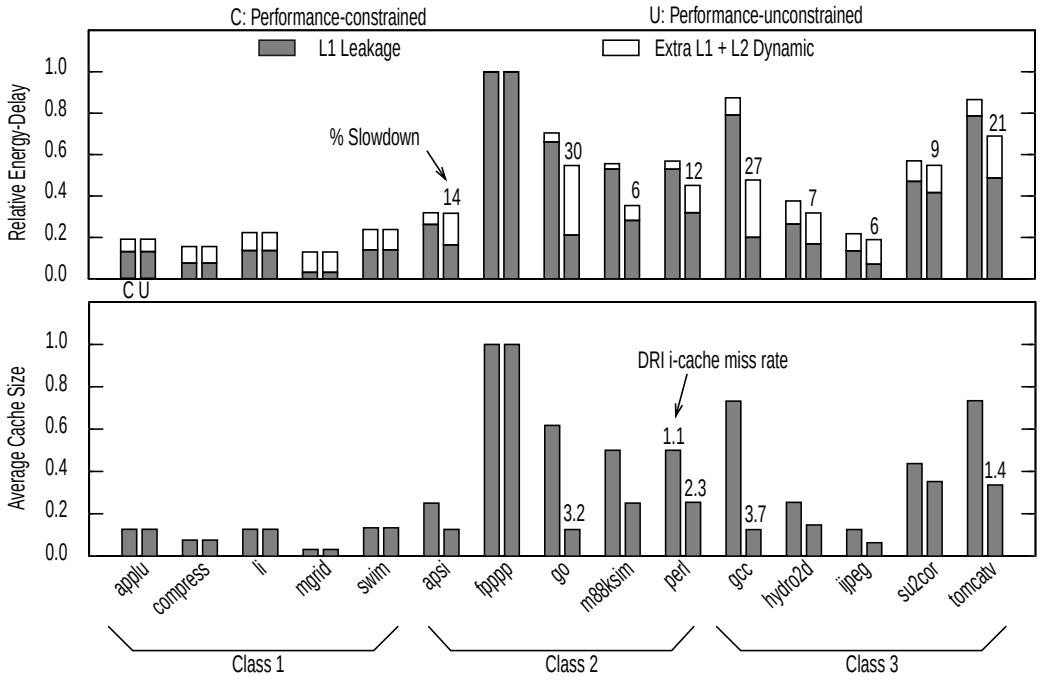

FIGURE 4: Base energy-delay and average cache size measurements.

DRI i-cache leakage energy numbers from Section 5.2 with the execution time.

Figure 4 shows our base energy-delay product and average cache size measurements normalized with respect to the conventional i-cache. The figure depicts measurements for both performance-constrained (left bars) and performance-unconstrained (right bars) cases. The top graph depicts the normalized energy-delay products. The graph shows the percentage increase in execution time relative to a conventional i-cache above the bars whenever performance degradation is more than 4% for the performance-unconstrained measurements. In the graph, the stacked bars show the breakdown between the leakage and the dynamic component due to the extra dynamic energy. The 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 performance-unconstrained case above the bars whenever the miss rates are higher than 1%.

From the top graph, we see that a DRI i-cache achieves large reductions in the energy-delay product as performance degradation is constrained, demonstrating the effectiveness of our adaptive resizing scheme. 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*. In *fpppp* the 64K i-cache is fully-utilized preventing the cache from resizing and reducing the energy-delay. The energy-delay products' dynamic component is small for all the benchmarks, indicating that both the extra L1 dynamic energy due to resizing bits is small and the extra L2 accesses are few, as discussed in Section 2.3.

There are only a few benchmarks (*gcc*, *go*, *m88ksim*, and *tomcatv*) which exhibit a significantly lower energy-delay under the performance-unconstrained scenario. For all these benchmarks, performance of the performance-unconstrained case is considerably worse than that of the conventional i-cache (e.g., *gcc* by 27%, *go* by 30%, *tomcatv* by 21%), indicating that the lower energy-delay product is achieved at the cost of lower performance.

From the bottom graph, we see that the average DRI i-cache size is significantly smaller than the conventional i-cache and the i-cache requirements largely vary across benchmarks. 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.*

The conventional i-cache miss rate (not shown) is less than 1% for all the benchmarks (highest being 0.7% for *perl*). The DRI i-cache miss rates are also all below 1%, except for *perl* at 1.1%, for the performance-constrained case. It follows that the absolute difference between DRI and conventional i-cache miss rates is less than 1%, well within the bounds necessary to keep the extra dynamic component low (computed in Section 5.2).

A DRI i-cache's simple adaptive scheme enables the cache to downsize while keeping a tight control over the miss rate and the extra L2 dynamic energy. Our miss rate measurements (not shown) for the performance-constrained experiments, where miss rate control is key, indicate that the largest absolute difference between the effective DRI i-cache miss rate and the miss-bound is 0.004 for *gcc*.

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. They mostly execute tight loops allowing a DRI i-cache to stay at the size-bound, causing the performance-constrained and performance-unconstrained cases to match. *Applu*, *compress*, *li*, *mgrid* and *swim* fall in this class, and primarily 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

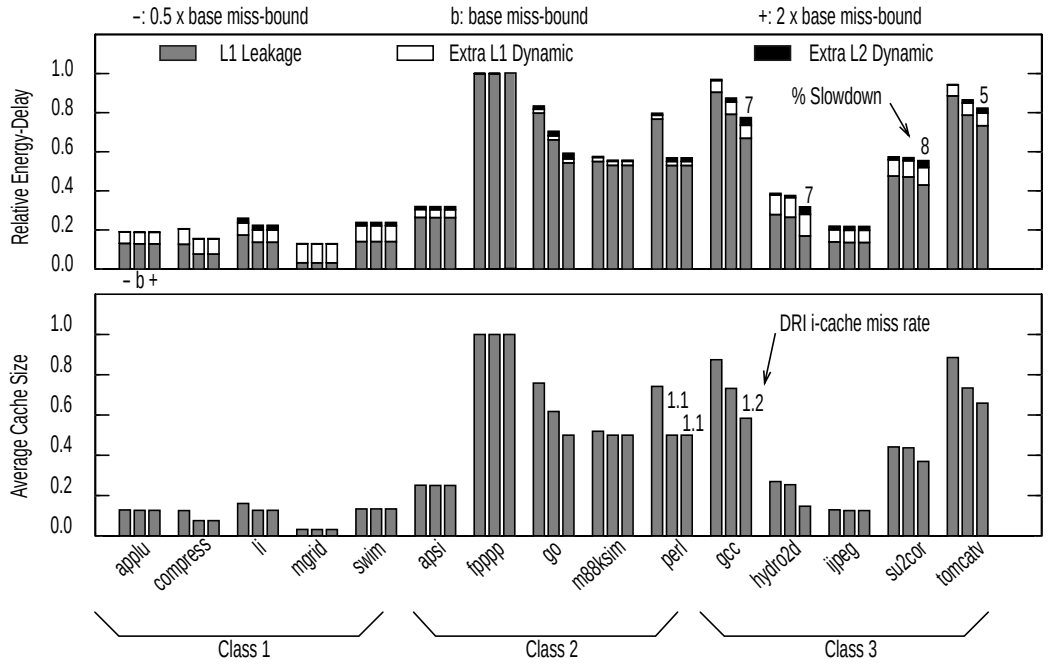

FIGURE 5: Impact of varying the miss-bound.

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 a significant performance loss. Consequently, the performance-constrained case 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 for this benchmark. Therefore, we disallow the cache from downsizing for *fpppp* by setting the size-bound to 64K. In the rest of the benchmarks, when performance is constrained, the dynamic energy overhead is much less than the leakage energy savings, allowing the cache to benefit from downsizing.

The last class of benchmarks exhibit distinct phases with diverse i-cache size requirements. *Gcc, hydro2d, ijpeg, su2cor* and *tomcatv* belong to this class of benchmarks. 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* both have relatively clear phase transitions. After the initialization phase requiring the full size of i-cache, these benchmarks consists mainly of small loops requiring only 2K of i-cache. 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' average sizes under both the performance-constrained and performance-unconstrained cases are relatively large.

5.3.1 Impact of Varying Miss-Bound

Figure 5 shows the results for varying the miss-bound to half and double the miss-bound for the base performance-constrained measurements, while keeping the size-bound the same. The top graph shows the effective energy-delay product normalized to the conventional i-cache leakage energy-delay, together with the percentage performance degradation for those cases which are higher than 4%. The bottom graph shows average cache sizes as a fraction of the conventional i-cache size, together with the miss rate for those 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 energy-delay products do not change significantly. Even when the miss-bound is doubled, the L1 miss rates stay within 1% and the extra L2 dynamic energy-delay does not increase much for most of the benchmarks. Therefore, 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 for more downsizing under higher miss-bounds. The bottom graph indicates that the DRI i-cache does not readily identify phase transitions in these benchmarks. These benchmarks achieve average i-cache sizes smaller than those of the base case, but incur between 5%-8% performance degradation compared to the conventional i-cache.

5.3.2 Impact of Varying Size-Bound

Figure 6 shows the results for varying the size-bound to double and half the size-bound for the base performance-constrained measurements, while keeping the miss-bound the same. *Fpppp's* base size-bound is 64K, and therefore there is no measurement corresponding to double the size-bound for *fpppp*. The top graph shows the effective energy-delay product normalized to the conventional i-cache leakage energy-delay and also the percentage slowdown for the cases which are higher than 4%. The bottom graph shows average

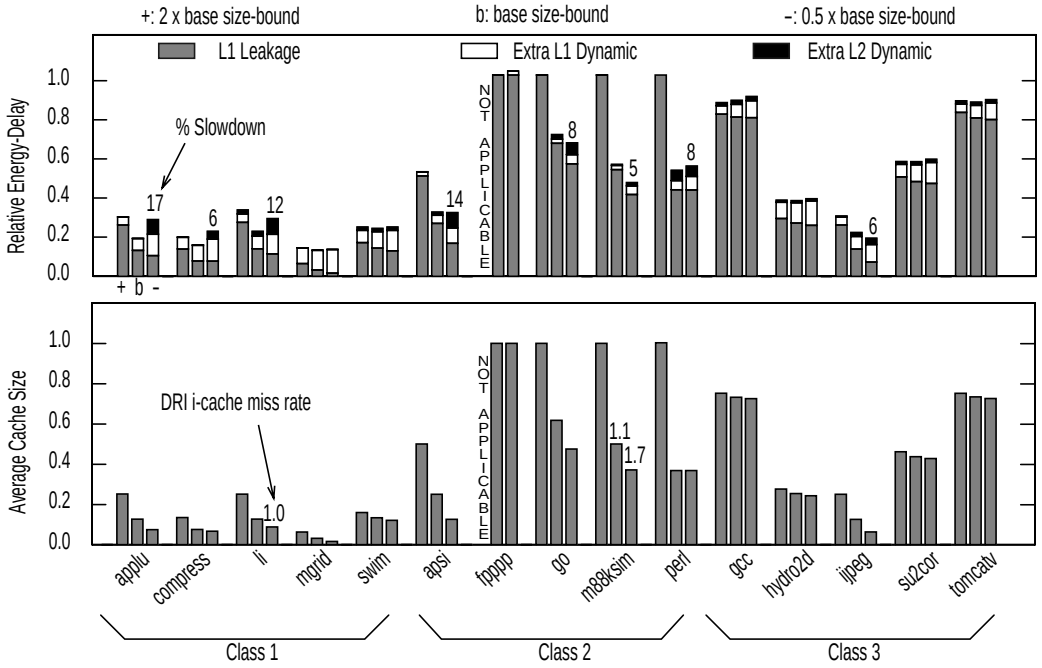

FIGURE 6: Impact of varying the size-bound.

cache sizes as a fraction of the conventional i-cache size, together with the miss rate for those cases which are above 1%.

The graphs show that a smaller size-bound results in a larger reduction in the average cache size, but the effect on the energy-delay varies depending on the benchmark class. The first class of benchmarks incur little performance degradation with the base 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. Therefore, doubling the size-bound simply increases the energy-delay and halving it increases the extra L2 dynamic energy, which worsens the energy-delay.

Decreasing the size-bound for the second class encourages downsizing at the cost of a lower performance due the benchmarks' large i-cache requirements. For the third class of benchmarks, the extra L1 dynamic energy incurred by decreasing the size-bound outstrips the leakage energy savings, resulting in an increase in energy-delay. *Fpppp*'s results for a 32K size-bound indicate that a poor choice of parameters may result in unnecessary resizing and actually increase the energy-delay beyond that of a conventional i-cache.

5.3.3 Impact of Varying Sense-Interval Length and Divisibility

In this section, we discuss our measurements varying the sense-interval length and divisibility. Ideally, we want the sense-interval length to correspond to program phases, allowing the cache to resize before entering a new phase. Our experiments show that a DRI i-cache is highly robust to the interval length for the benchmarks we studied. When varying the interval length from 250K to 4M i-cache accesses, the energy-delay product varies by less than 1% in all but one benchmark, and less than 5% in *go* due to its irregular phase transitions.

A large divisibility reduces the switching overhead in applications with frequent switching between two extreme i-cache sizes. Our experiments indicate that for all the benchmarks, a divisibility of four or eight (i.e., a factor of four or eight change in size) prohibitively increases the resizing granularity preventing the cache from assuming a size close to the required size, offsetting the gains from reduced switching overhead.

6 Conclusions

This paper explored an integrated architectural and circuit-level approach to reducing 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. Modern caches, however, are designed to meet the worst-case application demand, resulting in poor utilization and consequently high energy inefficiency in on-chip caches. 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 gated-V_{dd} to virtually eliminate leakage in the cache's unused sections.

We evaluated the energy savings and the energy performance trade-off of a DRI i-cache and presented detailed architectural and circuit-level simulation results. Our results indicated that: (i) There is a large variability in L1 i-cache utilization both *within* and *across* applications. A DRI i-cache effectively exploits this variability and reduces the average size of a 64K cache by 62% with performance degradation constrained within 4%; (ii) Lowering the cell threshold voltage from 0.4V to 0.2V results in doubling the cell speed and two orders of magnitude increase in leakage. A wide NMOS dual-V_t gated-V_{dd} transistor with a charge pump offers the best gated-V_{dd} implementation and virtually eliminates leakage with only 8% cell read time and 5% area increase; (iii) A DRI i-cache effectively integrates architectural and the gated-V_{dd} circuit techniques to reduce leakage in an L1 i-cache. A DRI i-cache reduces the leakage energy-delay product by 62% with performance degradation within 4%, and

by 67% with higher performance degradation; (iv) Our adaptive scheme gives a DRI i-cache tight control over the miss rate to keep it close to a preset value, enabling the DRI i-cache to contain both the performance degradation and the increase in lower cache levels' energy dissipation. Moreover, the scheme is robust and performs predictably without drastic reactions to varying the adaptivity parameters.

Acknowledgements

This research is supported in part by SRC under contract 2000-HJ-768. This material is also based upon work supported under a National Science Foundation Graduate Fellowship. We would like to thank Shekhar Borkar, Vivek De, Ali Keshavarzi, and Faith Hamzaoglu for information on leakage trends in cache hierarchies in emerging deep-submicron technologies.

References

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 32)*, pages 248–259, Nov. 1999.

[2] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic management techniques to reduce energy in high-performance processors. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 64–69, Aug. 1999.

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

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

[5] 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.

[6] 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.

[7] V. De. Private communication.

[8] I. Fukushi, R. Sasagawa, M. Hamaminato, T. Izawa, and S. Kawashima. A low-power SRAM using improved charge transfer sense. In *Proceedings of the 1998 International Symposium on VLSI Circuits*, pages 142–145, 1998.

[9] M. Hamada and 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.

[10] 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 2000 International Symposium on Low Power Electronics and Design (ISLPED)*, July 2000.

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

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

[13] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 30)*, pages 184–193, Dec. 1997.

[14] 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.

[15] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, P. C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 31(11):1703–1714, 1996.

[16] S. Mutoh, T. Douskei, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE Journal of Solid-State Circuits*, 30(8):847–854, 1995.

[17] 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.

[18] M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in cache memories. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 90–95, July 2000.

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

[20] Semiconductor Industry Association. The International Technology Roadmap for Semiconductors (ITRS). http://www.semichips.org, 1999.

[21] 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.

[22] L. Su, , R. Schulz, J. Adkisson, K. Byer, G. Biery, W. Cote, E. Crabb, D. Edelstein, J. Ellis-Monaghan, E. Eld, D. Foster, R. Gehres, and et. al. A high performance sub-0.25um CMOS technology with multiple thresholds and copper interconnects. In *IEEE Symposium on VLSI Technology*, 1998.

[23] 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.

[24] L. Wei, Z. Chen, M. 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.

[25] L. Wei, Z. Chen, and K. Roy. Double gate dynamic threshold voltages (DGDT) SOI MOSFETs for low power high performance designs. In *IEEE International SOI Conference*, pages 82–83, 1997.

[26] L. Wei and K. Roy. Design and optimization for low-leakage

with multiple threshold CMOS. In *IEEE Workshop on Power and Timing Modeling*, pages 3–7, Oct. 1998.

[27] 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.

[28] 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.