

CALCM Technical Report #2001-001



Evaluating Opportunity and Effectiveness of Cache Resizing to Reduce Energy Dissipation

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

June 2001

**Computer Architecture Lab at Carnegie Mellon (CALCM)
Department of Electrical and Computer Engineering**

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

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

Evaluating Opportunity and Effectiveness of Cache Resizing to Reduce Energy Dissipation

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

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

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

June 2001

Abstract

Cache memories are accounting for an increasing fraction of a chip's transistors and overall energy dissipation. Recent research advocates using "resizable" caches to exploit cache size variability in applications to reduce cache size and eliminate energy dissipation in the cache's unused sections with minimal impact on performance. Current proposals for resizable caches fundamentally vary in two design aspects: (1) cache organization, where one organization, referred to as *selective-ways*, varies the cache's set-associativity, while the other, referred to as *selective-sets*, varies the number of cache sets, and (2) resizing strategy, where one proposal *statically* sets the cache size prior to an application's execution, while the other allows for *dynamic* resizing both across and within applications. While the various design choices in these proposals vastly differ in how they exploit opportunity for cache resizing and their effectiveness in reducing energy, these choices have never been compared against each other.

In this paper, we evaluate the opportunity for and effectiveness of cache resizing in reducing energy dissipation. Rather than evaluating an exhaustive set of design choices, we select a *key* set of questions to understand the underlying application/system characteristics affecting design choices. We also propose and evaluate a hybrid resizable cache exploiting the advantages of both previous organizations. Using cycle-accurate performance and circuit simulation tools, we show that: (1) selective-ways and selective-sets perform comparably for i-caches, but selective-sets outperforms selective-ways for d-caches because selective-sets maintains set-associativity while reducing size benefitting applications with a high degree of conflict misses, (2) there is only a moderate degree of opportunity for dynamic resizing within applications, (3) a miss-ratio based dynamic resizing framework is more effective in capturing the opportunity for resizing within applications in i-caches than d-caches, because in d-caches misses are often overlapped making performance less sensitive to miss ratio, and (4) a hybrid organization of selective-ways-and-sets offers a richer spectrum of sizes than either individual organization, reducing the energy-delay by 36% and 32% in i- and d-caches as compared to selective-sets.

1 Introduction

The ever-increasing level of on-chip integration in CMOS technology has enabled phenomenal improvements in microprocessor performance but has also caused an increase in energy dissipation. High energy dissipation diminishes the utility of portable systems and reduces reliability, requires sophisticated cooling technology, and increases cost in all segments of the computing market including high-end servers [21]. 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 [16]. Therefore, a significant fraction of a chip’s energy dissipation is due to cache memories.

Current circuit techniques to reduce energy dissipation in caches typically trade off speed for lower energy dissipation in less performance-critical cache structures. Instead of solely relying on circuit techniques, recent research also advocates using “resizable” caches to reduce energy dissipation especially in high-performance caches [1,18]. Resizable caches are based on the observation that cache utilization varies *across* and *within* application execution. These caches allow hardware/software to customize the cache size to fit an application’s demands. By eliminating energy dissipation in the cache’s unused sections, resizable caches significantly improve energy-efficiency with *minimal* impact on application performance.

Current proposals for cache resizing fundamentally differ in cache organization, resizing framework, and how they exploit variability in applications’ cache utilization to save energy. One proposal [1] advocates a *selective-ways* cache organization which allows for varying the cache’s set-associativity. The other proposal [18] advocates *selective-sets* cache organization which allows for varying the number of cache sets. These cache organizations differ in (1) the offered range of cache sizes, (2) the offered resizing granularity — i.e., the distance between two adjacent offered sizes, (3) the allowable set-associativity at various resizings, and (4) the hardware complexity. The effectiveness of either cache organizations to reduce size and energy depends on the one hand on the application’s demand for a specific size and set-associativity and on the other hand the cache’s ability to meet the demands.

The two proposals also differ in “when” to perform cache resizing. The proposal for selective-ways [1] advocates “static” cache resizing by setting the cache size prior to an application’s execution, and exploits variation in cache utilization only *across* applications. The proposal for selective-sets [18] advocates a dynamic cache resizing framework based on monitoring cache miss ratio and resizes the cache to react to varying demand for cache size both *within* and *across* applications. The two resizing strategies differ in two respects (1) the ability to resize the cache during an application’s execution, and (2) the design complexity. The effectiveness of dynamic resizing depends on both the resizing opportunity within applications and the ability of the dynamic resizing mechanisms to seize the opportunity.

The previous studies on resizable caches focused on comparing each proposal with specific design choices (i.e., organization and strategy) against a conventional non-resizable cache. However, there exists no comparison between the two. This paper identifies the opportunity for cache resizing in a spectrum of applications, and draws and compares architectural choices from previous proposals to evaluate their effectiveness in capturing the opportunity. Rather than presenting an exhaustive set of results on all combinations of architectural choices for resizing, we select a *key* set of questions to help architects understand the underlying application/system characteristics affecting choice in cache resizing. We also propose and evaluate a hybrid selective-ways-and-sets resizable cache organization exploiting the advantages of both previous organizations.

Because circuit techniques are a viable alternative to reducing energy in less performance-critical cache designs with high access times, in this paper we only focus on *performance-critical cache designs* characteristic of L1/L2 caches in state-of-the-art processors. Moreover, we only consider resizable cache designs that have minimal (< 2%) to no impact on application performance relative to a non-resizable cache. We use cycle-accurate performance simulators and circuit energy and timing evaluation tools to perform our

evaluation. We present our evaluation in the form of answers to the following questions on key design choices:

- 1) Does selective-sets subsume selective-ways as a cache resizing technique?**
- 2) Is there opportunity for dynamic resizing?**
- 3) Does a miss-ratio based dynamic resizing framework capture the opportunity?**
- 4) How effective is a hybrid selective-ways-and-sets organization?**

Our results show that:

- Selective-ways reduces set-associativity and size at the same time, offers small cache sizes only at less set-associativity, and is less effective for applications that benefit from set-associativity. In 32K 4-way set-associative *instruction* caches (i-caches), selective-ways and selective-sets perform comparably reducing the energy-delay product on average by 53% and 59% relative to a non-resizable cache, respectively. However, in 32K 4-way set-associative *data* caches (d-caches), demand for set-associativity allows selective-sets to outperform selective-ways reducing the energy-delay product on average by 61% as compared to 48% relative to a non-resizable cache.
- Using an offline “oracle” analysis of cache access intervals, we estimate that dynamic resizing (within an application) could reduce the energy-delay product on average by 14% for i-caches and 18% for d-caches relative to a 32K 2-way static resizing selective-sets. Several applications exhibit a stable i-cache footprint size and no resizing opportunity. However, in applications with i-cache resizing opportunity, the footprint sizes vary significantly. Most applications exhibit some variation in d-cache footprint sizes and resizing opportunity. Overall, dynamic resizing opportunity is only moderate and may not justify sophisticated designs to capture the opportunity.
- In d-caches, performance is *not* always sensitive to large changes in miss ratio because out-of-order engines often overlap cache misses. A sharp increase in miss ratio, however, increases the access frequency to lower cache levels, increasing the energy dissipation and offsetting the gains from resizing. In contrast, in i-caches performance is directly related to miss ratio because i-cache misses in superscalar are not overlapped. Therefore, a miss-ratio based dynamic framework achieves energy-delays for d-caches and i-caches that are 12% and 6% higher than the offline estimate respectively.
- Selective-sets only offers cache sizes that are powers of two while selective-ways provides cache sizes with a linear resizing granularity. A hybrid selective-ways-and-sets organization offers a rich spectrum of cache sizes by taking advantage of the offered cache sizes in both organizations, reducing the energy-delay product relative to a 32K 4-way selective-sets by 36% in i-caches and 32% in d-caches.

The rest of the paper is organized as follows. Section 2, describes the source of dynamic energy dissipation in state-of-the-art cache memories. Section 3 presents our dynamic cache resizing framework and its application to resizable caches. Section 3.1 presents the energy models we use to estimate dynamic energy dissipation in caches. In Section 4 we describe the experimental methodology and results. Section 5 presents an overview of the related work. Finally, we conclude the paper in Section 6.

2 Resizable Caches

Resizable caches exploit the variability in cache size requirements in applications to save energy dissipation with minimal impact on performance. Resizable caches save energy by allowing portions of the cache to be enabled/disabled. To disable/enable cache sections, resizable caches exploit the partitioning of a cache into SRAM subarrays, found in modern high-performance implementations. To optimize for cache access speed, cache designers divide the array of blocks into multiple subarrays of SRAM cell rows, each containing one or more cache blocks [25]. Resizing electrically isolates cache sections in multiple subarrays to save energy [1]. We will describe the details of energy savings in Section 3.1.

There are two proposals for resizable cache architectures that fundamentally differ in how they exploit variability in the required cache size in applications [1,18]. Resizable caches primarily differ in two respects: (1) cache organization, dictating which cache dimensions are adjustable, and (2) resizing time, dictating when the caches readjusts these dimensions. At the organizational level, *selective-ways* offers adjustable set-associativity allowing the cache to eliminate energy dissipation in disabled cache ways. Alternatively, *selective-sets* offers an adjustable number of sets allowing the cache to eliminate energy dissipation in disabled sets. Similarly, *static resizing* allows resizing only across applications by adjusting size prior to an application’s execution. Alternatively, *dynamic resizing* exploits resizing both *across* and *within* application execution.

In the rest of this section, we first describe the base selective-ways and selective-sets cache organizations, and then propose a hybrid organization that enables adjustability in both cache set-associativity and the number of sets. Next, we describe the static and dynamic cache resizing frameworks.

2.1 Cache Resizing Organizations

There are two proposals for resizable cache organizations, which we call *selective-ways* and *selective-sets*. Selective-ways allows enabling/disabling entire associative ways in a cache. Figure 1 depicts the basic structure of a selective-ways resizable cache. As in conventional set-associative caches, at the higher level the data array is organized into cache ways. Each cache way consists of a number of subarrays. A *way-mask* allows disabling/enabling all the subarrays in a given way. Hardware or software can adjust the number of ways the cache uses by setting the way-mask. The cache access logic uses the way-mask to identify which cache ways to access.

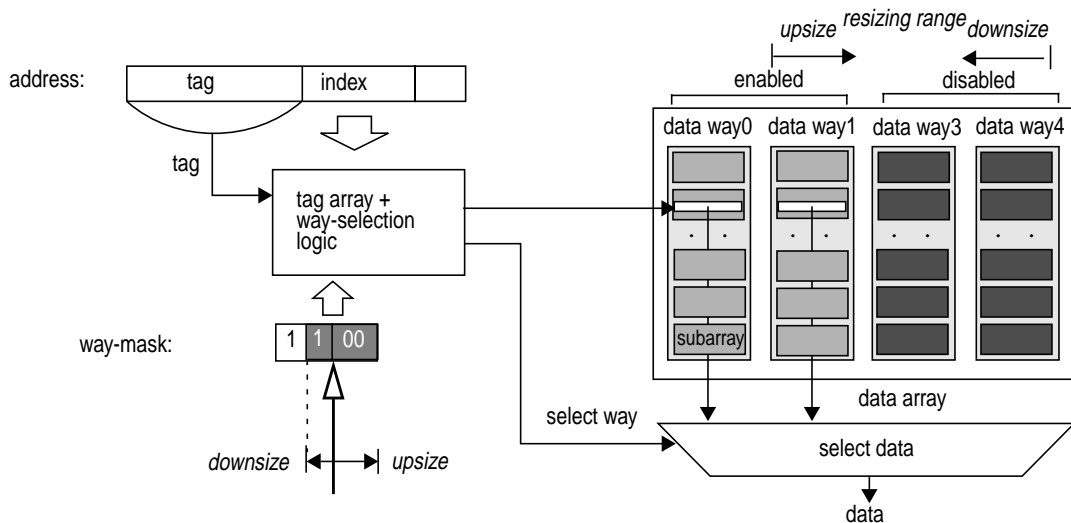


FIGURE 1: A selective-ways resizable cache organization.

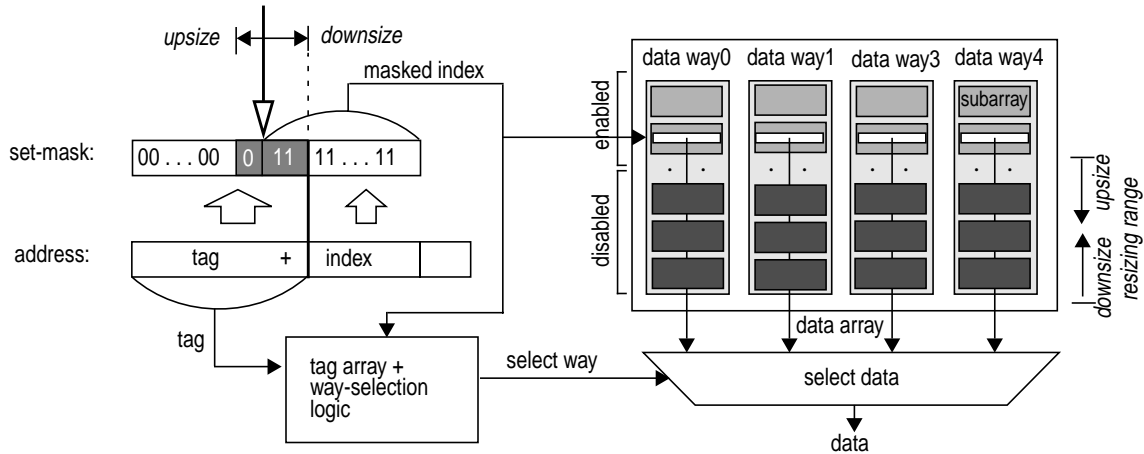


FIGURE 2: A selective-sets resizable cache organization.

Alternatively, selective-sets allows enabling/disabling cache sets. Figure 2 depicts the anatomy of a selective-sets cache organization. In a conventional cache, the number of cache sets and the cache block size dictate the set of *index* and *tag* bits used to look up a cache block. Therefore, changing the number of cache sets changes both the required index and tag bits. Selective-sets provides a *set-mask* to allow varying the number of cache sets and the used index bits. Because enabling/disabling occurs in multiples of subarrays (Section 3), the minimum number of sets achievable is a single subarray per cache way.

There are fundamental differences in these cache organizations in their complexity and effectiveness to offer the required cache size. First, the two organizations differ in applicability and the range of cache sizes offered. Selective-ways is advantageous because it changes size linearly in multiples of cache ways maintaining a constant resizing granularity. However, in high-performance caches (optimized for access time) which are often direct-mapped or use limited set-associativity, selective-ways is either not applicable or offers a small range of cache sizes. For instance, a 64K 2-way set-associative cache can only resize to a 32K direct-mapped cache using selective-ways, allowing at best 50% energy savings. Alternatively, selective-sets offers a large spectrum of cache sizes for low set-associative caches, is limited when set-associativity is high, and is not applicable to fully-associative caches. Moreover, selective-sets offered cache sizes are powers of two (because of the index-based set-mapping in conventional caches) allowing for fine-grain resizing only at smaller cache sizes but coarse-grain resizing at larger cache sizes. Therefore, selective-sets may be suboptimal when application working sets are large.

Moreover, selective-ways changes set-associativity along with size and may miss the significant opportunity for resizing for memory reference streams with small working sets but high conflict cache miss rates. Selective-sets maintains set-associativity while resizing increasing the opportunity for resizing for reference streams with high conflict miss rates.

A key advantage of selective-ways is its design simplicity. Selective-ways only requires an additional way-mask with corresponding logic. In contrast, selective-sets increases design complexity beyond the addition of a set-mask and its logic. Because resizing changes the number of tag bits, with smaller caches requiring a larger number of tag bits, selective-sets must use a tag array as large as that required by the smallest size offered. Therefore, using selective-sets, a cache of a given size requires a larger tag array which may be slower and dissipate more energy than a selective-ways cache of the same size and set-associativity. Moreover, selective-ways does not change the set-mapping of cache blocks and as such obviates the need for flushing blocks in the enabled subarrays upon resizing. In addition to flushing modified blocks prior to disabling arrays, selective-sets requires flushing all blocks (clean or modified) for which set-mappings change upon enabling subarrays.

Way size	Set-associativity			
	4-way	3-way	2-way	dm
8K	32K →	24K	16K	8K
4K	16K →	12K	8K	4K
2K	8K →	6K	4K	2K
1K	4K →	3K →	2K →	1K

Table 1: Enhanced resizing granularity using hybrid.

2.1.1 A Hybrid Organization

In this paper, we also propose and evaluate a hybrid organization for resizable caches. The key motivation behind a hybrid organization is that each of the resizable cache organizations offers a spectrum of cache sizes neither of which is a superset of the other. Selective-ways offers a spectrum of sizes that are multiples of a cache way size. Selective-sets offers a spectrum of sizes that are powers of two. A hybrid organization exploits the resizing granularity advantages of both organizations offering a richer spectrum of cache sizes than either organization alone, thereby optimizing energy savings by providing a size closest to the required demand for size by the application.

Table 1 illustrates cache size and set-associativities offered by a hybrid selective-ways-and-sets cache. For a base 32K 4-way set-associative cache and a subarray size of 1K, hybrid offers all of 32K, 24K, 16K, 12K, 8K, 6K, 4K, 3K, 2K, and 1K sizes. Whereas, a selective-ways cache would only offer 32K, 24K, 16K, and 8K sizes (indicated by the first row) and selective-sets cache would provide 32K, 16K, 8K, and 4K (indicated by the 4-way column). The table also depicts our simple resizing scheme. All the sizes between 32K to 3K simply go in steps between a 4-way and a 3-way configuration. For sizes less than 3K, the only configurations offered are those that further reduce set-associativity. This scheme follows the intuition that at higher cache sizes, capacity plays a bigger role than set-associativity while at lower cache sizes, set-associativity can significantly impact cache performance [12]. Downsizing from a 4-way 32K, our cache opts for a larger 24K size with a lower set-associativity of 3 ways rather than selecting a 4-way 16K cache as selective-sets would. Such an approach increases the opportunity for resizing for applications with working set sizes closer to 24K than 16K.

The table also indicates that there are redundant sizes (shaded gray in the table) offered by a hybrid cache. For instance, a hybrid cache can offer 8K as a size with any of 4-way, 2-way, and direct-mapped set-associativities. For redundant size, the hybrid cache offers the configuration with the highest set-associativity to minimize miss ratio and optimize the utilization of block frames.

2.2 Cache Resizing Opportunity

In this paper, we focus on high-performance resizable caches which optimize energy dissipation given a tight constraint on performance degradation with respect to a conventional (non-resizable) cache.¹ In such caches, opportunity for cache resizing primarily depends on three factors: (1) the available parallelism in an application’s (or application phase’s) cache accesses, (2) the available cache bandwidth and the system’s ability to overlap multiple cache miss latencies, and (3) the application’s (or application phase’s) primary cache working set size.

1. While reducing the cache size may also improve cache access speeds [2], such designs will require microarchitectural techniques that can exploit variability in cache access times which are beyond the scope of this paper.

Superscalar pipelines typically fetch instructions in program order and sequentially, exhibiting minimal parallelism in i-cache accesses [18]. Because, the instruction cache accesses are often on the critical path of execution, application performance is directly related to the i-cache's miss ratio. A resizable i-cache can exploit variability in instruction working set size to reduce cache size and save energy.

Unlike the instruction cache access stream, the data cache access stream in applications often exhibits parallelism due to independent sequences of computation requiring data cache accesses. In wide-issue superscalar engines with non-blocking data caches, when the application exhibits a high degree of memory parallelism, cache miss latencies can often be overlapped, removing the miss latency off the critical path of execution. A high degree of exploited parallelism in data cache accesses therefore enables aggressive downsizing with minimal impact on application performance.

Unfortunately, while aggressive downsizing may not decrease performance in non-blocking data caches, it may increase the overall energy dissipation in the system. Because downsizing increases the cache miss ratio, it also increases the frequency of accesses to the lower level caches. Caches typically exhibit sharp increases in miss ratio when a cache can no longer hold the primary working set of the application. Because a lower level cache is often an order of magnitude larger in size, a sharp increase in access frequency to a lower level cache may significantly increase the overall power dissipation offsetting the energy gains from resizing [1]. Therefore, opportunity for downsizing is also limited by the *extra* energy dissipated in the lower levels when the increase in miss ratio due to downsizing is high.

2.3 Cache Resizing Strategies

Cache resizing's effectiveness depends on the opportunity for resizing, the cache's ability to seize the opportunity for resizing, and the cache's offering of required sizes. Besides organization, a key design choice in resizing is "when" to resize. There are two proposals for resizing strategy. *Static resizing* allows resizing the cache prior to application execution, exploiting cache size variability across applications. Static resizing requires profiling an application's execution with different (static) cache sizes to determine the cache size with minimal energy dissipation given a tolerance for performance degradation. In static resizing, the application provides a cache size which the operating system loads into the cache's programmable size mask (i.e., the way-mask or the set-mask) prior to application's execution or upon a context switch.

Dynamic resizing reacts to application demand for resizing to customize cache size and optimize energy savings while an application is executing. Dynamic resizing uses extra hardware to monitor an application's execution and dynamically estimate performance and energy dissipation. When opportunity for resizing arises, dynamic resizing uses the cache size masks to resize the cache.

In this paper, we evaluate a simple miss-rate based dynamic resizing framework proposed in [18]. Hardware monitors the cache in fixed-length intervals measured in number of cache accesses. A miss counter counts the number of cache misses in each interval. At the end of each interval, hardware determines the need for cache upsizes/downsizes depending on whether the miss counter is higher/lower than a preset value, referred to as the *miss-bound*. To avoid thrashing, the framework prevents the cache from downsizing beyond a preset size, the *size-bound*. As in static resizing, the framework parameters are extracted offline through profiling.

Much like cache organization, there are fundamental differences in resizing strategy. Static resizing's key advantage is that it minimizes design complexity by fixing the size for the duration of an application's execution. In the absence of resizing opportunity within the application, static resizing obviates the need for hardware monitoring and may achieve optimal energy reduction. For instance, resizing opportunity within an application does not arise when the application exhibits a fixed working set size. Similarly, when performance degradation and the extra energy dissipation (due to increases in miss ratio) are *not* sensitive to vari-

ations in the working set size, there is no need for cache resizing. In such cases, static resizing may offer optimal energy savings if the cache configuration allows for a cache size close to the required size.

However, when there is opportunity for resizing within an application, static resizing fails to seize the opportunity and is suboptimal. For instance, an application with varying instruction cache footprint sizes across different phases will require different i-cache sizes while executing. In such a case, static resizing must provide an “in-between” size to satisfy the different program phases while limiting the performance degradation to the desired value.

In contrast, dynamic resizing may help seize resizing opportunity within an application and optimize the energy savings. Dynamic resizing, however, increases complexity and may require sophisticated hardware mechanisms to monitor and react to an application’s change in behavior. Dynamic resizing’s effectiveness in reducing energy depends on the accuracy and timeliness of the mechanisms to react to a change in application behavior. In general, online estimation of opportunity for resizing is difficult when performance degradation and energy dissipation are *not* sensitive to simple cache performance metrics such as miss ratio. Inaccurate resizing may result in both a large performance degradation and extra dissipated energy due to large increases in the miss ratio and accesses to the lower cache levels. Inaccurate resizing will at a minimum incur the performance and energy overhead of flushing some of the cache blocks in the disabled/enabled subarrays (Section 2.1). Furthermore, disabled subarrays may have included part of an application’s primary working set, resulting in an increase in the miss ratio to bring the blocks back into the new enabled subarrays.

A key advantage of dynamic resizing is that it may help overcome the limitations of coarse-grain resizing granularity specifically in selective-sets. Dynamic resizing can emulate an “in-between” cache size when the required cache size is in between two cache sizes offered by a resizable cache. The latter advantage diminishes for resizable caches with a finer-grain resizing such as our hybrid cache.

3 Energy Savings in Resizable Caches

Two categories of energy dissipation in deep-submicron CMOS caches are: “switching” energy and “leakage” energy. Switching energy is the energy dissipated in cache upon an access when the bitlines are charged/discharged [14,19]. Therefore, the switching is a function of the number of bit lines accessed. Leakage energy is the energy dissipated in all cache SRAM cells due to a “subthreshold” leakage current in transistors independent of the cache access frequency [19]. Previous studies on resizable caches only considered either switching energy dissipation [1] or leakage energy dissipation [18] but not both. This paper evaluates resizable cache’s effectiveness in reducing switching and leakage energy because both components are important in future designs [9]. In this section, we will first discuss the sources of energy dissipation and then explain how resizing together with circuit techniques effectively reduces energy dissipation.

To optimize for speed, cache designers divide the tag and data arrays into multiple subarrays of SRAM cell rows, each containing one or more cache blocks [11] (Section 2). Figure 3 depicts the anatomy of a cache subarray. In high-performance caches, *all* cache subarrays are precharged together prior to a cache access, to overlap the precharging time with the address decode and wordline assertion. Only a small number of subarrays (equal to the cache’s set-associativity) are actually accessed, leading to extremely high energy inefficiencies in modern caches [14]. While precharging *only* the required subarrays (i.e., using some of the address bits to identify the subarray to which the access goes) would reduce switching energy dissipation [10,17], it would increase the cache access time by as much as 30% (as per CACTI simulations [11]), diminishing the applicability of such an approach in high-performance designs. Instead, resizable caches select the appropriate cache size, disable the resulting unused subarrays (Figure 3), and reduce switching energy by precharging only the enabled subarrays.

Current technology scaling trends [4] require aggressively scaling down the threshold voltage to maintain transistor switching speeds. Unfortunately, subthreshold leakage current through transistors increases exponentially with decreasing threshold voltage, resulting in a significant amount of leakage energy dissipation at a low threshold voltage. Many circuit techniques allow trading off speed for reduced leakage in less performance-critical circuits. For instance, multiple threshold voltage processes offer higher threshold transistors for caches with reduced leakage but slower access times. Instead, resizable caches can exploit transistor stacking — also known as gated- V_{dd} [18] — to allow aggressive threshold-voltage scaling and still prevent inordinate leakage. Gated- V_{dd} places a single wide transistor in series with a cache block in a subarray (Figure 3). The self reverse-biasing of stacked transistors reduces leakage current by orders of magnitude [26]. By selecting the appropriate cache size, resizable caches can turn off the gated- V_{dd} for the disabled subarrays, virtually eliminating leakage in the disabled subarrays.

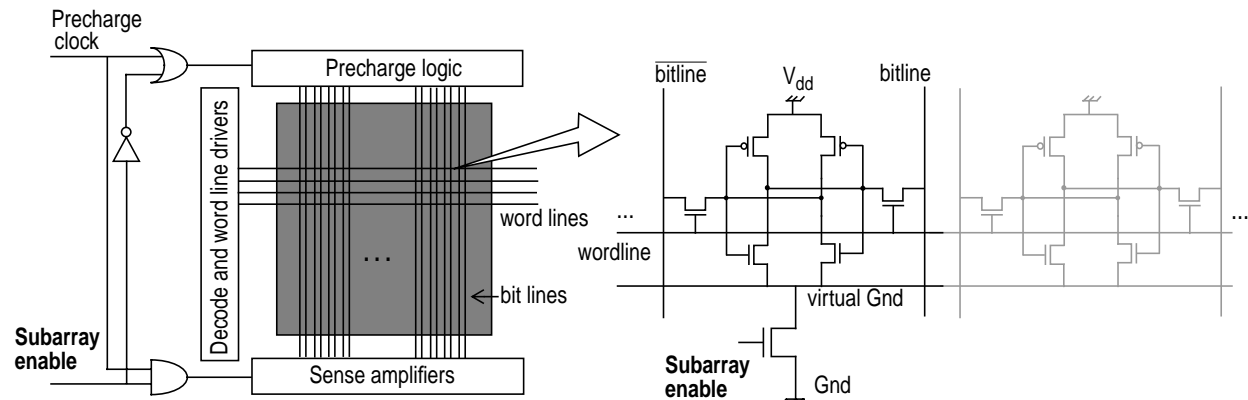


FIGURE 3: Energy reduction circuit techniques.

3.1 Energy Considerations in Resizable Caches

A resizable cache eliminates energy dissipated in the disabled subarrays but increases L2 energy due to extra L1 misses (discussed in Section 2), regardless of the organization or the resizing strategy used. The two key factors in the extra L2 energy are the energy per L2 access and the increase in the number of L2 accesses (or extra L1 misses). Because L2 accesses are not as critical as L1 accesses, delayed precharging is a viable option to reduce the first factor, and even high-performance chips such as the Alpha 21264 employ this option for L2 [11]. Because extra L1 misses affect both performance and energy, resizable caches control the second factor disallowing downsizings that cause numerous extra L1 misses.

In addition to the extra L2 energy component, there are other components specific to the organization and resizing strategy (also discussed in Section 2). For selective sets, L1 energy increases due to the extra resizing tag bits. This increase is insignificant because the number of resizing tag bits is small (usually between 1 and 4) compared to the number of bit lines in a cache block (e.g., 256). Irrespective of the organization, dynamic resizing needs to flush cache blocks between sense intervals, incurring extra energy dissipation in both L1 and L2. Because sense interval lengths are in millions of cycles and the total number of L1 cache blocks is in thousands, the flush overhead is easily amortized over the interval, even in the worst case when all the cache blocks are flushed.

4 Results

In this section, we present the answers and supporting results for each question asked in the introduction. We use SimpleScalar 3.0 to run SPEC applications using reference inputs. We use *ammp*, *vortex*, and *vpr*, from SPEC2000 and nine other applications from SPEC95. For *gcc*, *jpeg*, *m88ksim*, *su2cor*, and *tomcatv* we run the entire application; for the other applications we skip one billion instructions and run the next two billion instructions to reduce simulation turnaround time. Table 2 shows our system configuration parameters.

We compute cache switching energy [14] and leakage energy [18], assuming a 0.18 micron technology. We choose an aggressive supply voltage of 1.0V to enable high performance. Our 32K 4-way cache has a power rating of 1.1W, of which 39% is leakage. Our power rating is higher than that of StrongARM or similar processors that do not focus on performance because we assume an aggressive, high-performance design. However, our results hold qualitatively for high-performance cache design points irrespective of their specific leakage components and power ratings.

We use the energy-delay product to present the results because it is a well-established metric used in low-power research. The energy-delay product ensures that both reduction in energy and accompanying performance degradation are taken into account together. For each design point, the relative energy-delay is obtained by normalizing its energy-delay with respect to that of the non-resizable cache with the same size and set-associativity. We always present the lowest energy-delay product achieved for each benchmark within a performance degradation constraint of 2% compared to a non-resizable cache. We assume a base cache of 32K with 1K subarrays. Other configuration parameters are specified as they are varied in each section.

4.1 Does Selective-sets subsume selective-ways as a cache resizing technique?

In this section, we compare two cache resizing organizations, selective-ways and selective-sets. Based on our discussion in Section 2, we expect selective-sets to achieve the best relative energy-delay for high-performance designs (1) by providing smaller minimum sizes for the applications requiring small cache size and (2) by maintaining set-associativity for the applications having a sizable amount of conflict misses. Applications requiring finer cache sizes around the maximum size and having a small number of conflict misses can benefit from selective-ways. Therefore, for those applications, selective-ways is expected to perform as well as or better than selective-sets. Because applications' instruction reference streams are usually less conflict-miss intensive than their data reference streams, selective-ways in i-caches is expected to be able to perform well enough to be comparable to selective-sets.

Figure 5 compares the relative energy-delays of static selective-ways and selective-sets for 32K 4-way d-caches and i-caches. We use a 4-way set-associative cache for the base case because it provides a reasonable granularity for selective-ways and is a feasible set-associativity for high-performance systems. For d-caches, selective-ways reduces the relative energy-delay by an average of 48%, and selective-sets reduces

Instruction issue & decode	8 issues per cycle	Base L1 i-cache	32K, 2-way; 1 cycle latency
Reorder buffer entries	128	Base L1 d-cache	32K, 2-way; 1 cycle latency
LSQ size	128	L2 unified cache	1M, 4-way; 12 cycle latency
Branch predictor	2-level hybrid	Memory access latency	80 cycles + 4cycles per 8 bytes
Write-back buffer entries	8	Number of mshrs	8

Table 2: Base system configuration parameters.

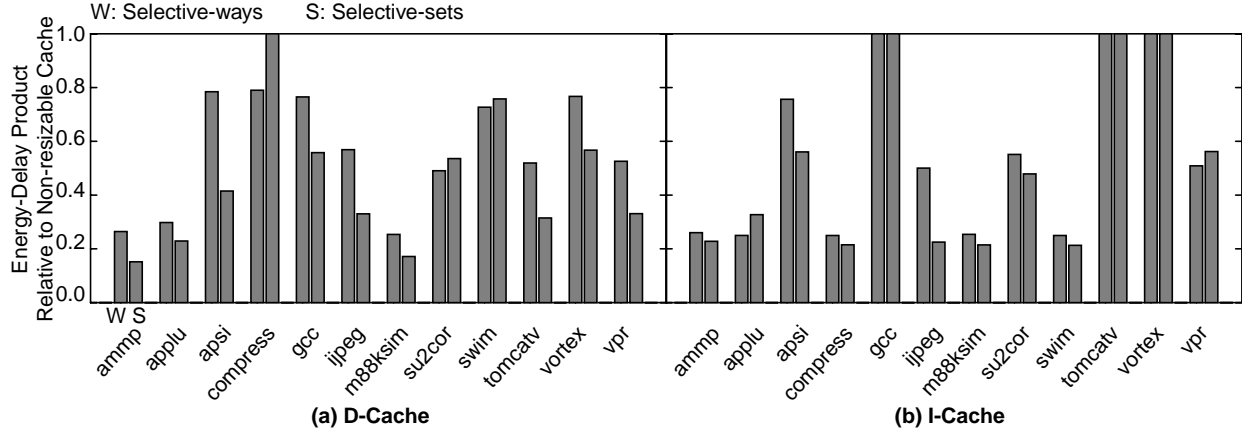


FIGURE 4: Relative energy-delay for 4-way static selective-sets and selective-ways.

relative energy-delay by 61%. For i-caches, the numbers are 53% and 59%, respectively. For d-caches, selective-sets averages 13% better energy-delay than selective-ways. The difference is 6% for i-caches, indicating that selective-sets subsumes selective-ways for both cache types. The gap is larger in d-caches because conflict misses in low set-associativity caches prevent selective-ways from downsizing in many applications. Also, large working sets prevent some applications from experiencing any i-cache downsizing.

In d-caches, for nine benchmarks out of twelve, selective-sets shows better energy-delay reduction than selective-ways. Six applications, *apsi*, *gcc*, *jpeg*, *tomcatv*, *vortex* and *vpr*, mainly benefit from selective-sets' ability to maintain set-associativity and prevent conflicts. Three benchmarks, *ammp*, *applu* and *m88ksim*, require small cache sizes and take advantage of the smaller minimum size (4K) provided by selective-sets. For the remaining applications, *compress*, *su2cor* and *swim*, selective-ways shows better energy-delay reduction than selective-sets. *Compress* requires granularity at large cache sizes which is provided by selective-ways but not by selective-sets. For *su2cor* and *swim*, selective-ways and selective-sets use the same cache size. However, because lower set-associative caches read fewer subarrays and dissipate less L1 energy on each access than higher set-associative caches, selective-sets dissipates more energy than selective-ways at the same size. *Swim*'s relative energy-delay is also impacted by a large amount of extra L2 energy.

For i-caches, *ammp*, *compress*, *m88ksim*, and *swim* require small cache sizes throughout execution and take advantage of the small minimum size available in selective-sets. *Ijpeg*, *apsi*, and *su2cor* require set-associativity rather than cache size to keep the performance in the allowable range. Therefore, selective-sets results in better energy-delay reduction for these benchmarks. *Vpr* and *applu* exhibit the same behavior as *su2cor* and *swim* for d-caches: both selective-ways and selective-sets choose the same cache size, but selective-ways dissipates less energy. *Gcc*, *vortex*, and *tomcatv* have no cache downsizing because their working sets are larger than 32K and downsizing incurs large performance degradation.

When cache access time is less critical, higher set-associativity caches are used. To illustrate the differences in resizing behavior for high set-associativity caches, Figure 5 depicts relative energy-delay for selective-ways and selective-sets for 8-way, 32K caches. For 8-way caches, selective-ways provides a broader range of resizing opportunities than selective sets, and our results confirm that selective-ways outperforms selective-sets in most cases. The exceptions, *su2cor*, *swim*, *tomcatv*, *vortex*, and *vpr* d-caches, are due to conflict misses and performance degradation when selective-ways downsizes below a 4-way cache. For caches in which access time is less critical, circuit techniques are also applicable to reduce energy dissipation. In the remainder of this paper we focus strictly on high-performance designs and use a 2-way set-associative cache for our base case.

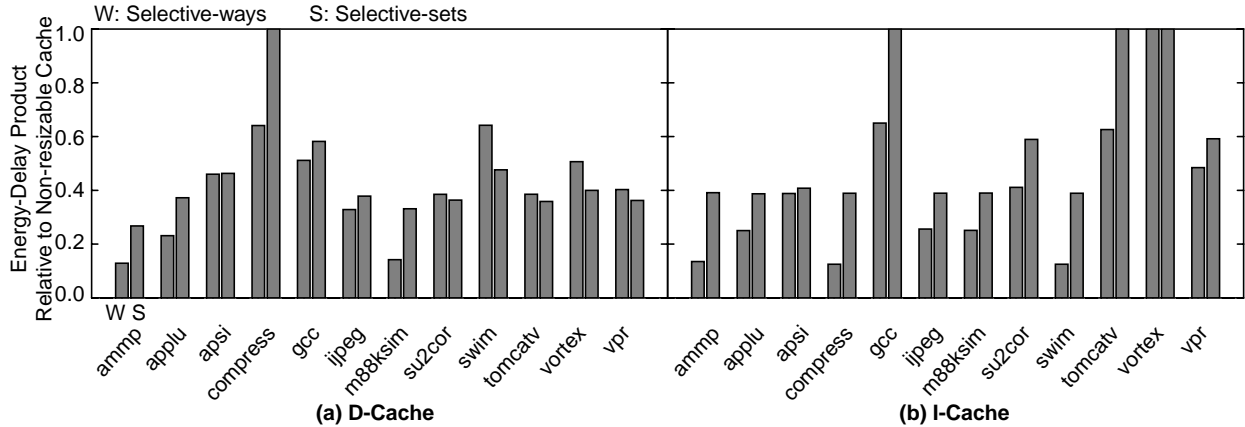


FIGURE 5: Relative energy-delay for 8-way static selective-sets and selective-ways.

Figure 6 shows relative energy-delay for selective-ways and selective-sets for 2-way, 32K caches. The results for selective-sets are fundamentally the same as with 4-way caches in Figure 4, although *ammp* and *m88ksim* are now able to downsize their d-caches further because a smaller minimum size of 2K is available. For selective-sets, the average energy-delay reductions are 63% and 65% for d- and i- caches respectively. For selective-ways, the average relative-energy delay reductions of 31% for d-caches and 30% for i-caches are much smaller than the reductions for selective-sets because selective-ways resizing is restricted to a minimum size of 16K. We have shown that selective-sets subsumes selective-ways for high-performance caches with set-associativities as high as 4. We therefore choose 2-way selective-sets as our base case.

4.2 Is there opportunity for dynamic resizing?

In this section we examine the effects of application behavior on resizing opportunity. Resizing opportunity exists because of variations in working set size both within and across applications. Though the concept of resizing opportunity is straightforward, the requirement to identify and analyze each phase of program execution makes determining the amount of available resizing opportunity difficult. We use a technique called *offline analysis* to estimate the resizing opportunity in each application. Comparisons between the results from offline analysis and static resizing indicate resizing opportunity that static resizing does not exploit.

We develop offline analysis to estimate cache resizing opportunity for applications. We divide each application’s execution into intervals made up of one million non-speculative cache accesses (i.e., instruction fetches for i-caches and committed loads/stores for d-caches) and analyze each interval to determine potential resizing. Additional experiments not shown indicate that the analysis is not sensitive to interval lengths within an order of magnitude of 1 million accesses. We compute the energy and latency of each interval for all cache sizes available using selective-sets. Our goal is to identify the best cache size to choose on a per interval basis. Initially we choose the cache size with the lowest energy-delay for each interval; however for most applications this initial choice will not meet our performance constraint of 2%. Therefore we iteratively upsize the cache for the specific intervals which save the most clock cycles while increasing energy the least. When the total number of execution cycles is within the performance constraint, we stop upsizing. While offline analysis does not take into account resizing overhead that may occur due to increased resizing, it provides an estimate of potential resizing opportunity using performance and energy information for the entire execution. Although the results from offline analysis are not strictly optimal, they provide estimates of the available opportunity.

Figure 7 (a) shows the ratio of energy-delay for 32K 2-way d-cache offline analysis to that of static selective-sets. Figure 7 (b) shows the corresponding i-cache results. For d-caches, offline analysis estimates an average relative energy-delay that is 82% that of static resizing. For i-caches, offline analysis estimates an

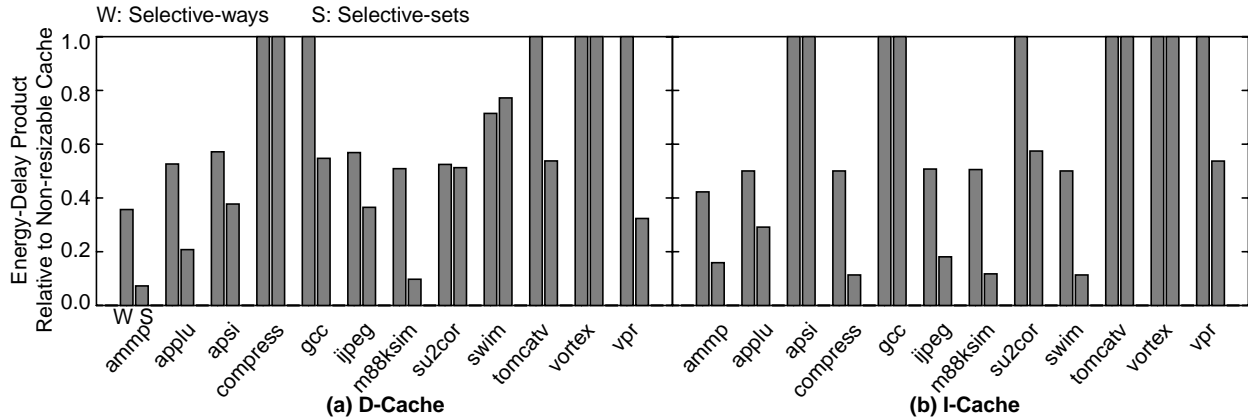


FIGURE 6: Relative energy-delay for 2-way static selective-sets and selective-ways.

average that is 86% that of static resizing. The figures indicate that several applications have significant resizing opportunity not exploited by static resizing. When static resizing does not offer the cache size required by the application or working set sizes vary during execution, opportunity for additional resizing exists.

To further analyze resizing opportunity, Figure 8 presents cache size profiles for offline analysis results in Figure 7. The x-axis of each graph represents intervals used in analysis, and the y-axis represents the cache size in K for the interval. The top twelve graphs are for d-caches and the bottom twelve for i-caches. The curves have different values and many are not flat, showing that variability in cache size demand exists. None of the curves are flat at 32K, which indicates each application has some opportunity for resizing

We observe three types of behavior and resizing opportunity within the application profiles. The simplest is those that have a constant size throughout execution. This indicates opportunity for resizing across applications but not within applications. Constant size requirements can be captured by static resizing. For d-caches, *m88ksim* and *ammp* exhibit constant sizes. For i-caches, *compress*, *m88ksim*, and *swim* exhibit constant sizes. The next type of behavior, which cannot be captured by static resizing, is variation in working set size within the application, indicated by changes in cache resizing behavior over many intervals. An example of working-set variation in d-caches is *compress*, which shows three distinct phases. *Applu*, *jpeg*, and *su2cor* are examples of periodic variation in d-cache working set size as execution phases repeat. *Applu* and *jpeg* also indicate periodic variation in i-cache working-set.

The third behavior observed in the results is unavailable-size emulation, which occurs when the cache size required by the application is not offered. The resizable cache chooses sizes above and below the required size to achieve emulation. Unavailable-size emulation occurs because there is too much performance or energy degradation at a smaller size but little degradation at a larger size. Static resizing cannot capture unavailable-size emulation because of the performance degradation at the smaller size. Dynamic resizing may be able to amortize the degradation by spending some time at the larger size. Additional sizes might also be captured by using a hybrid selective-sets/selective-ways resizing scheme.

While unavailable-size emulation may be difficult to distinguish from working-set variation based on the curves of Figure 8, some examples are apparent. Examples include *vortex* and *gcc* for i-caches, which experience unavailable-size emulation between 32K and 16K. The oscillation within the three distinct d-cache working set phases of *compress* indicates that this application experiences unavailable-size emulation between 32K and 16K as well as between 16K and 8K in the middle phase. For i-caches, *jpeg* exhibits periodic unavailable-size emulation between 4K and 2K within the periods requiring a 4K working set. The ranges of behaviors and sizes indicate significant opportunity for dynamic resizing.

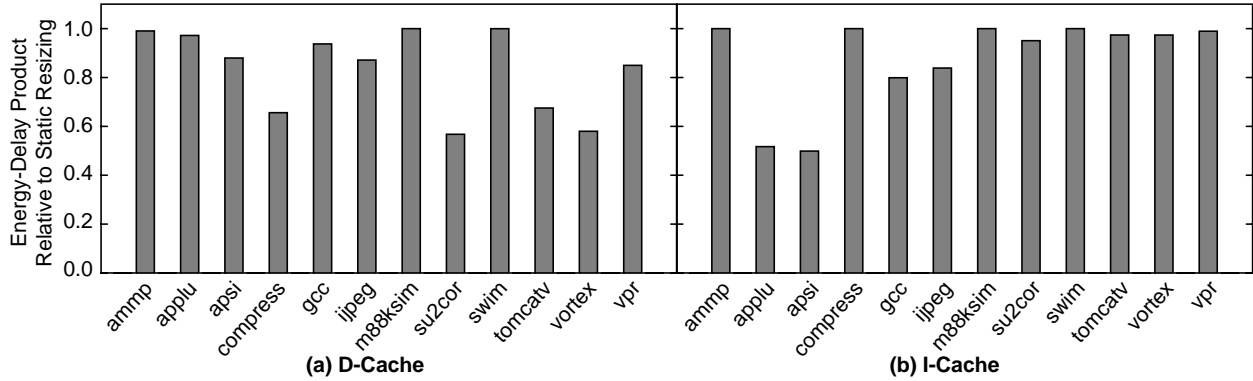


FIGURE 7: Energy-delay for offline analysis relative to static selective-sets.

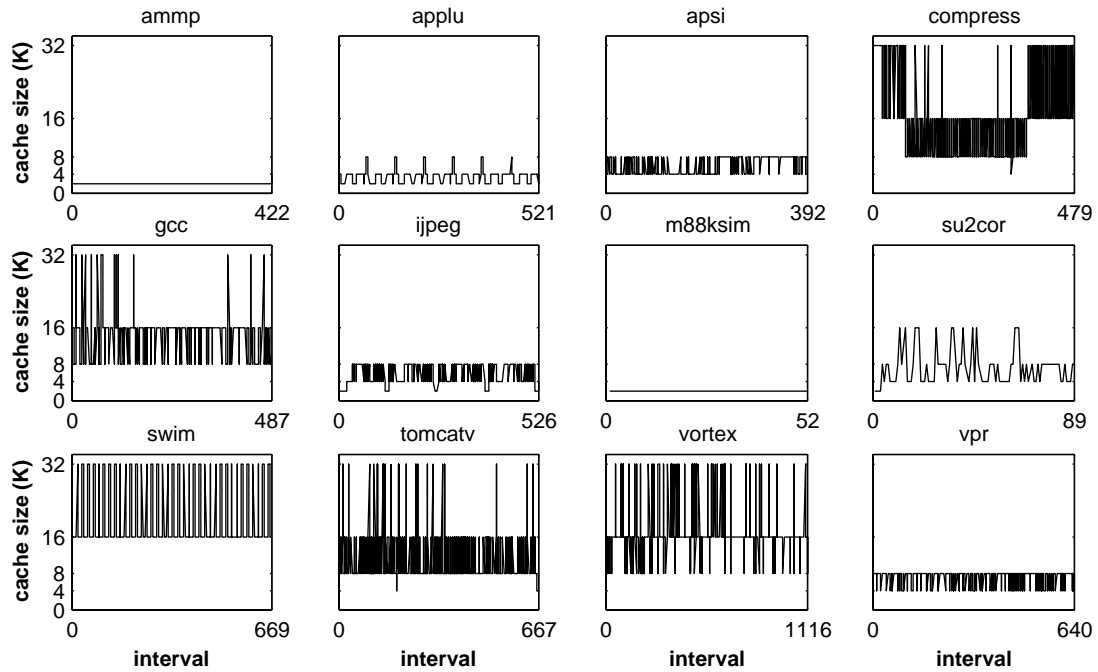
4.3 Does a miss-ratio based dynamic resizing framework capture resizing opportunity?

In this section, we investigate how a miss-ratio based dynamic resizing strategy proposed in [18] exploits the opportunity for dynamic resizing shown in Section 4.2 and what system characteristics influence the effectiveness of the strategy. When miss latency is exposed to the critical path by application or system characteristics, additional misses significantly degrade performance and discourage further cache downsizing. We can use miss-ratio to control cache downsizing without impacting performance if we can capture variability in misses that are exposed to the critical path. First we evaluate the effectiveness of miss-ratio based dynamic resizing, and then we examine how system characteristics impact the effectiveness this dynamic resizing strategy.

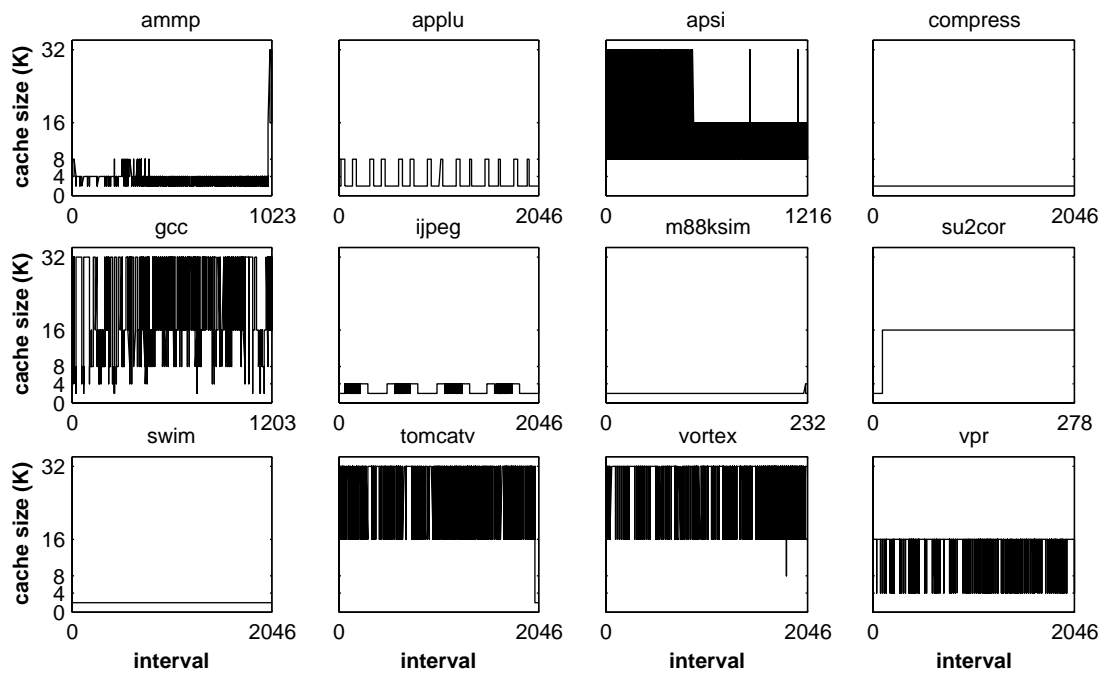
4.3.1 Effectiveness of miss-ratio based dynamic resizing

Figure 9 compares miss-ratio based dynamic resizing and offline analysis to static selective-sets. For miss-ratio based dynamic resizing, the resizing interval length is fixed to one million committed cache accesses. Miss-bound and size-bound are determined on a per application basis to achieve the best energy-delay product for the application. For i-caches, dynamic resizing shows better energy-delay reduction than static resizing for the applications in which offline analysis shows significant resizing opportunity beyond the static resizing results. For d-caches, there are eight applications showing resizing opportunity beyond static, but dynamic resizing exploits the opportunity for only three of them, *compress*, *su2cor*, and *vortex*. For applications where dynamic resizing always chooses the same size as static resizing, the energy-delay for dynamic resizing may be worse because dynamic resizing must adapt to the required size during execution. Due to many applications showing little or no additional resizing opportunity, the average energy-delay ratio between dynamic and static resizing is small, 94% for d-caches and 93% for i-caches. The values for d-caches and i-caches are 12% and 7% larger than the offline analysis results reported in Section 4.2, indicating miss-ratio based dynamic resizing is more successful at capturing additional opportunity in i-caches.

Because i-cache accesses in superscalar pipelines are typically sequential and on the critical path of execution, i-cache miss-ratio strongly impacts application performance. Miss-ratios can successfully be used as an indicator of performance changes from i-cache resizing because a small increase in i-cache miss-ratio can significantly degrade performance. Unlike i-cache accesses, d-cache accesses typically exhibit significant instruction-level parallelism depending on application and system characteristics. When d-cache miss latency can be overlapped, for example, by a wide-issue superscalar engine with non-blocking data caches, d-cache accesses are less critical to performance. Therefore, d-cache resizing is not as sensitive to miss-ratio as i-cache resizing, and large miss-ratios may be acceptable in d-cache resizing when they do not degrade performance beyond the constraint. However, the energy from a large number of extra L2 accesses



(a) D-Cache



(b) I-Cache

FIGURE 8: Offline analysis cache size profiles for d-caches (top graphs) and i-caches (bottom graphs).

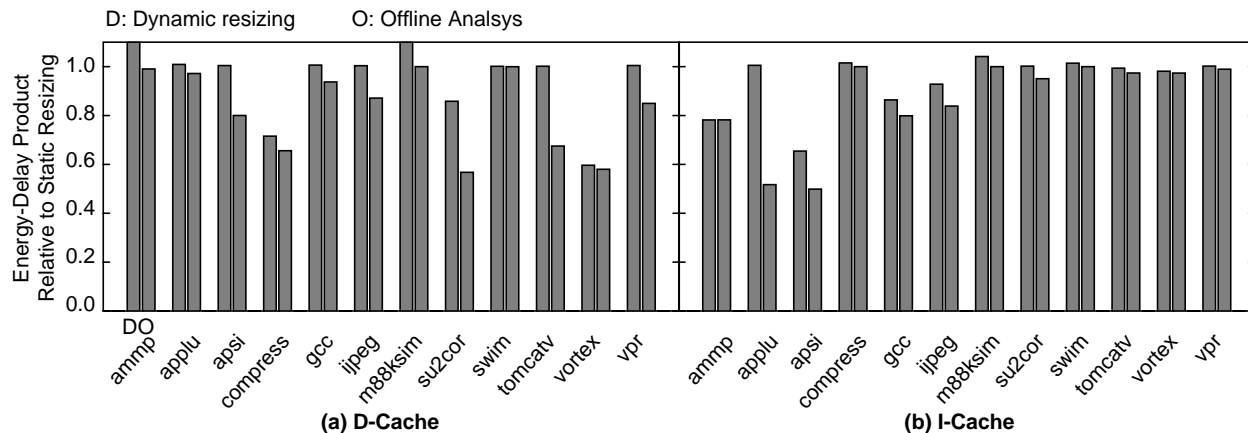


FIGURE 9: Energy-delay for dynamic resizing and offline relative to static resizing.

may also constrain resizing opportunity even if the misses do not affect performance as mentioned in Section 2.2.

In Table 3, we show the increase in miss-ratio between a non-resizable cache and a static selective-sets cache, or *delta miss-ratio*, as an indication of an application’s tolerance and sensitivity for additional L1 misses in resizing. Delta miss-ratio represents the miss-ratio increase in the application that can occur with a selective-sets cache size that does not degrade performance by more than 2%. We show results for both 2-way and 4-way caches to confirm that the behavior is consistent across set-associativities.

Regardless of set-associativity, delta miss-ratios for i-caches are small and less than 1%. For applications which experience d-cache resizing, however, the delta miss-ratio varies widely across the applications and is often greater than 1%. The applications for which delta miss-ratio is small for d-caches, such as *gcc*, *su2cor*, and *tomcatv* for 2-way caches, are sensitive to misses below the achieved size.

Our results in Table 3 show that i-cache resizing is much more sensitive to cache miss-ratio than d-cache resizing. The sensitivity to misses explains why miss-ratio based dynamic resizing is most effective in i-caches. The results imply that the overlaps in cache accesses and miss-ratio play a key role in determining the effectiveness of miss-ratio based cache resizing. For applications that can tolerate many additional overlapped misses, miss-ratio does not tightly control resizing.

4.3.2 Effect of system characteristics on miss-ratio based dynamic resizing

To show the effects of reducing overlap in misses, we present results for d-caches with high L2 latency and with low L1-L2 bandwidth in Figure 10 (a) and Figure 11 respectively. We simulate the former by increasing the L2 access latency from 12 to 18 cycles and the latter by reducing the number of miss state holding registers (mshrs) from 8 to 1. We use these results to examine the effects of making d-cache accesses more critical to performance

For d-caches with high L2 latency, the ratio between the energy-delays of dynamic and static resizing is 83%. For low-bandwidth d-caches, the corresponding ratio is 82%. This compares to 94% in the base-case from Section 4.3.1, indicating that dynamic resizing is more effective at capturing opportunity in high-

	ammp	applu	apsi	compress	gcc	ijpeg	m88ksim	su2cor	swim	tomcatv	vortex	vpr	ammp	applu	apsi	compress	gcc	ijpeg	m88ksim	su2cor	swim	tomcatv	vortex	vpr
2-way	2.7	1.6	2.8	0.0	0.8	3.1	1.0	0.0	7.4	0.2	0.0	2.1	0.2	0.0	0.0	0.0	0.0	0.1	0.0	0.5	0.0	0.0	0.0	0.1
4-way	0.6	1.3	3.0	0.0	0.7	0.8	0.6	0.0	6.7	0.2	0.8	1.5	0.5	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1

Table 3: Delta-miss ratio (%) for 2-way and 4-way static selective-sets.

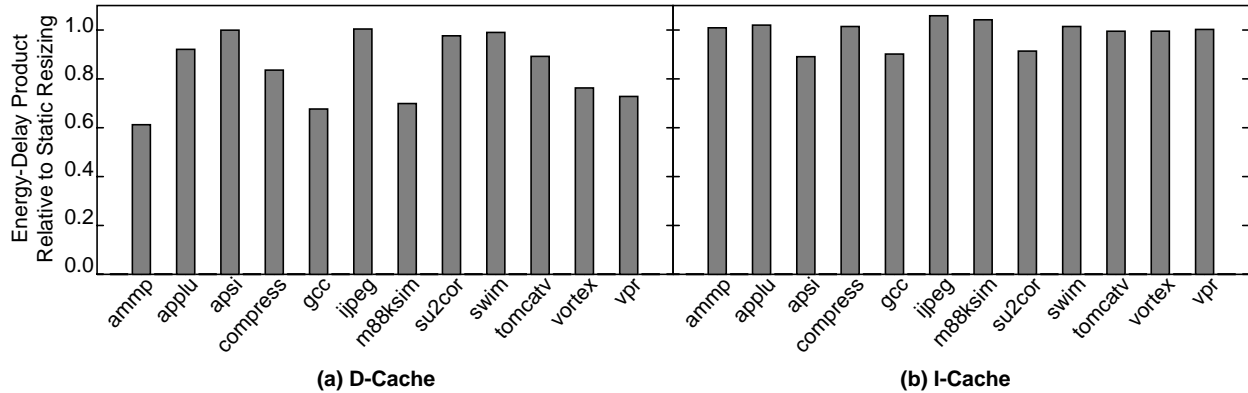


FIGURE 10: Energy-delay with 18-cycle L2 latency for dynamic resizing relative to static resizing.

latency and low-bandwidth situations. Static resizing fails to resize at all for *compress*, *gcc*, *swim*, and *vortex* in the d-caches with high L2 latency and for *compress*, *gcc*, *swim*, *tomcatv*, and *vortex* in the low-bandwidth d-caches. This is expected because of the larger performance impact of misses which is less favorable to cache downsizing than in the base case. However, dynamic resizing still squeezes savings out of applications with d-cache misses that are exposed to the critical path.

One indication of memory exposure to the critical path is IPC; a higher IPC indicates more potential parallelism and less exposure. For applications with high IPC (greater than 2.0) in the base case, specifically *applu*, *apsi*, *jpeg*, and *swim*, dynamic resizing does not achieve substantially lower energy-delay than static-resizing, because these applications exploit large amounts of parallelism in d-cache accesses. In the other lower IPC applications, the ratio of dynamic resizing energy-delay to that of static resizing is an average of 76%, significantly smaller than the 94% value for the base case with low L2 latency.

Low-bandwidth between L1 and L2 shows similar results with several notable exceptions. For one high IPC application, *jpeg*, static resizing could not downsize at all because the application's d-cache access pattern requires large bandwidth between L1 and L2. However, removing parallelism in the bandwidth allows dynamic resizing effectively utilizes some of the available resizing opportunity. *Su2cor* and *vpr* are low IPC applications for which we would expect dynamic resizing to outperform static resizing. However, their performance degradations are near 2% in the base case and become larger as bandwidth decreases, preventing additional dynamic resizing.

We also examine results for i-caches with high L2 latency as seen in Figure 10 (b). We do not examine an i-cache with 1 mshr because our i-cache is already blocking and low-bandwidth. For i-caches with high L2 latency, the results are similar to the base case but downsizing is discouraged everywhere due to the larger performance impact of misses. Despite the overall reduction in downsizing potential, dynamic resizing still manages to capture some opportunity for energy savings.

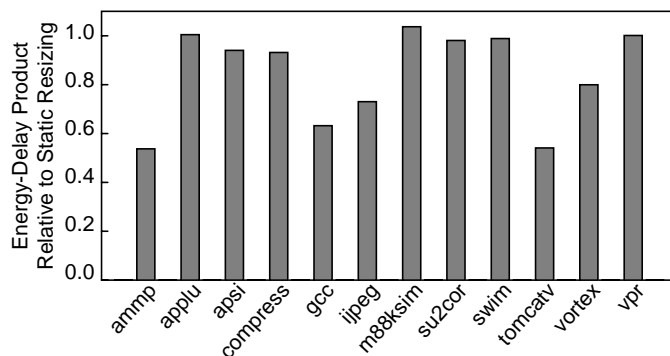


FIGURE 11: Energy-delay for low L1-L2 bandwidth dynamic resizing relative to static resizing.

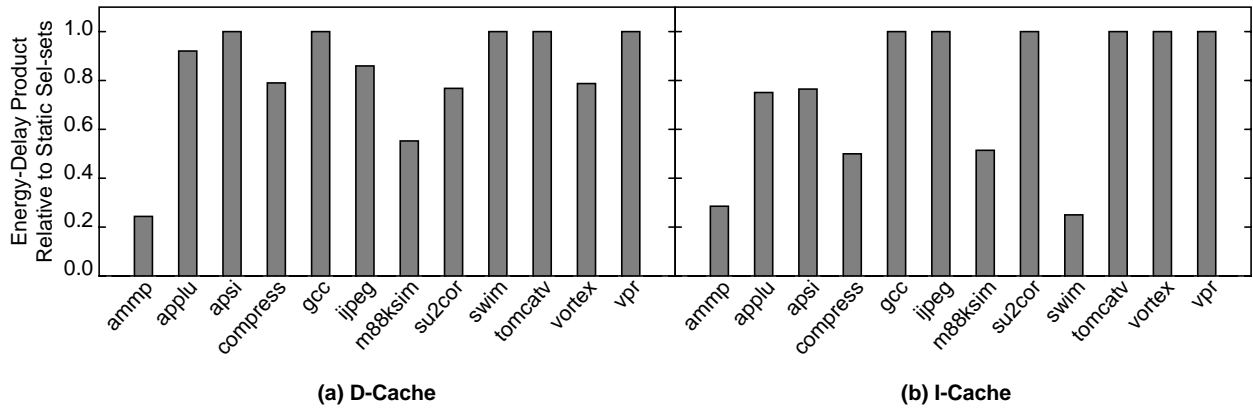


FIGURE 12: Energy-delay for static hybrid resizing relative to static selective-sets.

4.4 How effective is a hybrid selective-ways-and-selective-sets organization?

In this section, we investigate and evaluate hybrid organization. The hybrid organization captures and enhances the primary advantages of selective-ways and selective-sets by offering a richer spectrum of cache sizes. Hybrid offers better resizing granularity than either of selective-ways and selective-sets and therefore improves optimization by providing sizes closer to the actual cache size demand of the applications. Also, hybrid organization offers smaller sizes for applications demanding small cache sizes.

Figure 12 presents the energy-delays for static hybrid resizing for d- and i-caches relative to static selective-sets. The results are based on a 32K 4-way set-associative cache, in which hybrid organization has the resizing granularity described in Table 1. We do not include selective-ways in the figures because we saw in Section 4.1 that selective-sets generally subsumes selective-ways. The results show that hybrid organization achieves equal or better energy-delay reduction than selective-sets without exception. On average, hybrid’s energy delay is 64% of the result of static selective-sets for d-caches and 68% of the result of static selective-sets for i-caches.

As we forecasted, there are two situations for which hybrid organization does better than the selective-sets. For *compress*, *jpeg*, *vortex*, *su2cor* in d-cache and *compress*, *m88ksim*, *ammp*, *swim*, *apsi* in i-cache, its ability to offer better resizing granularity plays a role and show better energy-delay reduction than selective-sets. The cache sizes utilized by these applications in the hybrid organization are not supported by selective-sets. Smaller sizes offered by hybrid which are less than the minimum size of selective-sets or selective-ways are utilized in *m88ksim*, *ammp* for d-cache, *compress*, *m88ksim*, *ammp*, *swim* for i-cache.

5 Related Work

A number of previous studies have focused on architectural/microarchitectural techniques to reduce the energy dissipation in modern processors. Manne, et al., [16] propose pipeline gating to reduce energy dissipation due to mispredicting branches. Brooks, et al., propose and/or evaluate several techniques to reduce energy dissipation including narrow-width operands [5], and loop unrolling and memoing [6]. Toburen, et al., [23] evaluate instruction scheduling techniques to reduce energy dissipation. Vijaykrishnan, et al., [24] evaluate the impact of several performance-enhancing architectural and compiler techniques on energy dissipation and power.

There are a number of studies that have specifically focused on reducing the energy dissipation in cache memories. Many of these propose placing small energy-efficient buffers (e.g., 128-256 bytes in size) in front of caches to filter traffic to the cache. Examples include block buffers [7,22], filter cache [15], and the loop cache [3]. Filtering, however, is only effective when application working sets are extremely small and is otherwise not applicable. Moreover, when working sets do not fit, filtering increases the critical path access time for accessing cached information and may significantly reduce performance. Others propose way-prediction for selective precharging data arrays in set-associative caches [13]. Unfortunately, way-predictors increase the cache access critical path, do not achieve high prediction accuracies, and may reduce overall performance [8] and increase the energy-delay product. Moreover, way-prediction tables contribute to energy dissipation.

Recently, there have been three proposals for cache resizing [1,18,20] two of which focus on reducing energy dissipation. Ranganathan, et al. [20], propose a statically resizing selective-ways i-cache and use it to partition the cache and use the unused part as auxiliary storage for instruction reuse information to increase performance. Albonesi [1] proposes a statically resizing a selective-ways cache, and evaluates the cache's effectiveness in reducing switching energy. Powell, et al. [18], propose a dynamically resizing selective-sets i-cache, and evaluate its effectiveness to reduce leakage energy dissipation. In this paper, we draw *key* resizing architectural design aspects from Albonesi's and Powell, et al.'s proposals, to evaluate effectiveness of and opportunity for cache resizing to reduce energy dissipation. We also consider overall energy dissipation including both switching and leakage energy, and propose a hybrid cache organization and that exploits advantages of both selective-ways and selective-sets.

6 Conclusions

Recent research advocates using resizable caches to reduce energy dissipation in cache memories with minimal impact on performance. There are two proposals for cache resizing that vastly differ in their effectiveness in exploiting opportunity for cache resizing in applications. The previous proposals for resizable caches fundamentally vary in two design aspects: (1) cache organization, where one organization, referred to as *selective-ways*, varies the cache's set-associativity, while the other, referred to as *selective-sets*, varies the number of cache sets, and (2) resizing strategy, where one proposal *statically* sets the cache size prior to an application's execution, while the other allows for *dynamic* resizing both across and within applications. While the various design choices in these proposals vastly differ in how they exploit opportunity for cache resizing and their effectiveness in reducing energy, these choices have never been compared against each other.

To date, there are no previous studies comparing the merits of the various design choices for cache resizing in the previous proposals. Rather than evaluating an exhaustive set of design choices, in this paper we selected a *key* set of questions to understand the underlying application/system characteristics affecting choice in cache resizing.

- 1) Does selective-sets subsume selective-ways as a cache resizing technique?**
- 2) Is there opportunity for dynamic resizing?**
- 3) Does a miss-ratio based dynamic resizing framework capture the opportunity?**
- 4) How effective is a hybrid selective-ways-and-sets organization?**

Using cycle-accurate performance and circuit simulation tools, we showed that: (1) selective-ways and selective-sets perform comparably for i-caches, but selective-sets outperforms selective-ways for d-caches because selective-sets maintains set-associativity while reducing size benefitting applications with a high degree of conflict misses, (2) there is only a moderate degree of opportunity for dynamic resizing within applications, (3) a miss-ratio based dynamic resizing framework is more effective in capturing the opportunity for resizing within applications in i-caches than d-caches, because in d-caches misses are often overlapped making performance less sensitive to miss ratio, and (4) our hybrid organization of selective-ways-and-sets offers a richer spectrum of sizes than either individual organization, significantly improving the energy-delay product.

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] R. Balasubramonian, D. Albonesi, A. Buyutosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 33)*, Dec. 2000.
- [3] 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.
- [4] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.
- [5] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 13–22, Jan. 1999.
- [6] D. Brooks, V. Tiwari, and M. Martonosi. Watch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.
- [7] J. Bunda, W. Athas, and D. Fussell. Evaluating power implications of CMOS microprocessor design decisions. In *Proceedings of the 1994 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 147–152, Apr. 1994.
- [8] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *Proceedings of the Second IEEE Symposium on High-Performance Computer Architecture*, Feb. 1996.
- [9] V. De and S. Borkar. Technology and design challenges for low power and high performance. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 163–168, 1999.
- [10] J. H. Edmondson, P. I. Rubinfeld, P. J. Bannon, B. J. Benschneider, D. Bernstein, R. W. Castelino, E. M. Cooper, D. E. Dever, D. R. Donchin, T. C. Fischer, A. K. Jain, S. Mehta, J. E. Meyer, R. P. Preston, V. Rajagopalan, C. Somanathan, S. A. Taylor, and G. M. Wolrich. Internal organization of the Alpha 21164, a 300-MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal*, 7(1), 1995.
- [11] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, 10(14), 1996.
- [12] M. D. Hill. A case for direct-mapped caches. *IEEE Computer*, 21(12):25–40, Dec. 1988.
- [13] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 273–275, Aug. 1999.
- [14] M. B. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. pages 143–148, Aug. 1997.
- [15] 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.
- [16] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.
- [17] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoepfner, 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.
- [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] P. Ranganathan, S. Adve, and N. P. Jouppi. Reconfigurable caches and their application to media processing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 214–224, June 2000.
- [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] C.-L. Su and A. M. Despain. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of the 1995 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 63–68, 1995.

- [23] M. C. Toburen, T. M. Conte, and M. Reilly. Instruction scheduling for low power dissipation in high performance microprocessors. In *Proceedings of the 1998 Power Driven Microarchitecture Workshop*, pages 14–19, June 1998.
- [24] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106, June 2000.
- [25] 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.
- [26] 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.