Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches

Se-Hyun Yang and Babak Falsafi
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
{sehyun, babak}@cmu.edu
http://www.ece.cmu.edu/~powertap

Abstract

High-performance caches statically pull up the bitlines in all cache subarrays to optimize cache access latency. Unfortunately, such an architecture results in a significant waste of energy in nanoscale CMOS implementations due to high leakage and bitline discharge in the unaccessed subarrays. Recent research advocates bitline isolation *to control precharging of individual subarrays using bitline precharge devices. In this paper, we carefully evaluate the energy and performance trade-offs of bitline isolation, and propose a technique to exploit nearly its full potential to eliminate discharge and reduce overall energy in level-one caches.*

Cycle-accurate and circuit simulation results of a wide-issue superscalar processor indicate that: (1) in future CMOS technologies (e.g., 70nm and beyond), cache architectures that exploit bitline isolation can eliminate up to 90% of the bitline discharge, (2) on-demand precharging (i.e., decoding the address and subsequently precharging the accessed subarrays) is not viable in level-one caches because precharging increases the cache access latency, and (3) our proposal for gated precharging *to exploit subarray reference locality and precharging only the recently accessed subarrays eliminates nearly all of bitline discharge in nanoscale CMOS caches with only a 1% of performance degradation.*

1 Introduction

High-performance level-one (L1) caches increasingly account for a significant fraction of energy dissipation in wide-issue out-of-order processors [4,22]. To reduce the bitline capacitive load and achieve faster access times, these caches are divided into multiple subarrays of SRAM cell rows. To hide bitline precharging time and minimize cache access latency, these caches typically pull up the bitlines in all subarrays statically or on every clock cycle [5]. Unfortunately, such aggressive and blind bitline precharging results in a significant energy discharge through the bitlines even in unaccessed subarrays. The energy waste is exacerbated by: (1) the increasing leakage in recent CMOS technologies [3], and (2) the trend towards using highly-ported caches (e.g., data caches in superscalar and data/instruction caches in SMT) which employ multiple bitlines for an SRAM cell column.

Recent proposals [8,22] advocate *bitline isolation* as a technique to reduce energy discharge through the bitlines in L1 caches. Bitline isolation turns off the precharge devices located between bitlines and the processor's supply voltage to avoid pulling up bitlines for cache cells that are unlikely to be accessed in the near future. Controlling the bitlines of individual subarrays, however, requires both accurate and timely architectural mechanisms that predicts when to turn on the bitlines; inaccurate or late precharging may adversely impact cache access time, program execution time, and overall energy dissipation. Moreover, frequent switching of precharge devices may dissipate significant amounts of energy because these devices tend to be large, offsetting gains from bitline isolation.

Unfortunately, prior proposals either apply bitline isolation infrequently [1,22] (e.g., once per million instructions, over a group of subarrays) to amortize the performance/energy overhead of bitline isolation over large overall energy savings, or tacitly assume there is little overhead associated with bitline isolation [8]. These proposals do not evaluate or exploit the full potential for energy savings using bitline isolation.

In this paper, we carefully quantify the performance/energy trade-offs of bitline isolation and propose architectural techniques necessary to realize the full potential of bitline isolation in nanoscale CMOS L1 caches. Based on cycle-accurate architectural simulations, and timing and energy analysis from circuit-level simula-

Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36 2003)
0-7695-2043-X/03 $17.00 © 2003 **IEEE**

tions for a wide-issue out-of-order superscalar with a subset of SPEC2000 and Olden benchmarks, we show that:

- **Energy overhead:** The energy overhead of bitline isolation in the past/current CMOS technologies is so large (e.g., 200% in 180nm) that it nearly offsets the energy savings achieved by isolating bitlines. Fortunately, the overhead is decreasing as technology scales and will be insignificant in the future beyond 70nm technology. This result suggests that bitline isolation can be applied more aggressively in the future.
- **Potential savings:** By using an oracle that identifies accessed subarrays with no delay, we quantify the potential savings of bitline isolation in future CMOS technologies. For 70nm, the oracle reduces bitline discharge in data and instruction caches on average by 89% and 90% respectively, corresponding to 46% and 41% of the cache energy saving opportunities.
- **Precharging timeliness:** On-demand precharging using information from the address to identify the accessed subarrays on demand is not viable because it increases the cache access latency. Our results indicate that the increased L1 cache access latency degrades performance by 9% in data caches and 7% in instruction caches.
- **Gated precharging:** The cache's subarray references exhibit high locality with most cache accesses within a given execution window occurring in a small number of hot subarrays. We propose *gated precharging* to effectively exploit cache subarray locality and achieve near-optimal bitline precharging by capturing the potential. Gated precharging in 70nm reduces 83% and 87% of the bitline discharge and 42% and 36% of the overall energy dissipation from data and instruction caches, respectively, with only a 1% performance degradation.

The rest of the paper is organized as follows. Section 2 presents bitline precharging mechanisms and the bitline isolation technique. Section 3 briefly describes the experimental setup used throughout this paper. In Section 4, we look into the energy implication and the potential energy savings of bitline isolation. Section 5 and Section 6 present architectural techniques that exploit the potential of bitline isolation. We look into on-demand subarray precharging and propose gated precharging based on subarray reference locality. Section 7 presents the related work. Section 8 concludes the paper.

2 Background: Bitline Precharging & Isolation

Figure 1 depicts a typical 6-T SRAM cell with precharge devices. A read operation begins with the two bitlines precharged to the supply voltage. The address is supplied to the decode logic which activates the selected row's wordline. The SRAM cells on the row read their values out onto the precharged bitlines and establish a small voltage differential by slightly discharging one of the bitlines. The sense amps recognize this differential in the two bitlines and buffer the values for subsequent consumption by the pipeline. After a read, the voltage on the two bitlines must be equalized by precharging them to the processor's supply voltage.

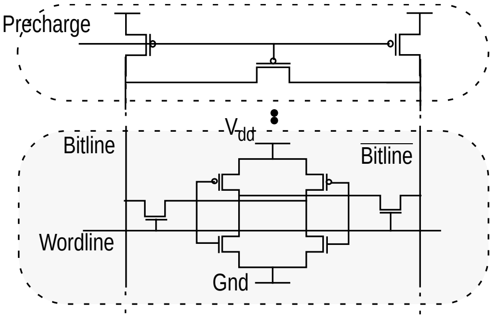


FIGURE 1: 6-T SRAM cell and precharge logic.

Bitline precharging is achieved through either *clocked precharging* that clocks the precharge devices every cycle or *static pull-up* that statically leaves them on all the time [5]. Static pull-up has the advantage that it does not require a heavily loaded precharge clock signal. Moreover, clocked precharging requires precise timing on the precharge clock which is often difficult to engineer. Therefore, recent designs [7] advocate static pull-up for the performance-critical L1 caches. We assume static pull-up for the base cache configuration in this paper.

To optimize the cache access speed, cache designers for modern high-performance caches segment bitlines and divide the array of SRAM cell rows into multiple subarrays [19]. To further improve clock speed and cache access latency, these caches overlap bitline precharging with other cache operations, such as address decoding or output driving, and hide the entire bitline precharging latency of each cache access. As such, the caches blindly precharge all the subarrays, irrespective of the accessed subarray.

In the past, this blind precharging scheme has been a viable approach to reduce the cache access latency without a significant cost. However, the large and growing subthreshold leakage in future CMOS technologies incurs a large bitline discharge from all statically pulled-up bitlines, even in subarrays that are not accessed. This bitline discharge is different from the one caused by a cell read. Current architectural trends towards using multiple ports (e.g., data caches in superscalar and data/instruction caches in SMT) requiring multiple bitlines for an SRAM cell column further exacerbate the large bitline discharge. We measure this bitline discharge to be 76% of the overall

Table 1: Circuit parameters.

Feature size (nm)	180	130	100	70
Supply voltage (V)	1.8	1.5	1.2	1.0
Clock frequency (GHz)	2.0	2.7	3.5	5.0

leakage dissipation in dual-ported SRAM cells. The large bitline discharge results in a significant energy waste.

The energy waste due to blind precharging can be reduced by determining which subarrays will be accessed and precharging only these subarrays. The precharge devices in the other subarrays are turned off, isolating their bitlines from the supply voltage, and turned back on prior to an access. Turning off the precharge devices cuts off the bitline leakage paths between the supply voltage and bitlines and reduces the discharge. We refer to this technique as *bitline isolation*.

Unfortunately, there are a number of key challenges the cache designers must overcome to fully exploit bitline isolation. First, the energy overhead of switching precharge devices may be high enough to offset the overall energy reduction if the precharge devices are switched frequently. Second, the cache requires an accurate mechanism to identify the subarray to be precharged prior to an access. Finally, depending on the precharging latency, the subarray must be identified and precharged early in the pipeline to allow precharging to overlap with the cache access and avoid an increase in overall access latency.

The first application of bitline isolation appeared in the Alpha 21164's L2 cache [2,6] as an extension of clock gating. This cache predecodes the address and subsequently turns on the precharge devices only for the relevant subarrays. While the precharge device switching energy is potentially large in L2 subarrays, this overhead is offset by the large energy savings from reducing the capacitive load on the L2's heavily-loaded clock distribution. Moreover, the increase in access latency due to delayed precharging is amortized over the L2's long overall access latency.

Recently, other researchers have applied bitline isolation to performance-critical L1 caches. Leakage-biased bitlines [8] do not carefully address the energy and performance overhead of bitline isolation. Resizable caches [1,22] predict the demand for cache size, and select the corresponding group of subarrays for static pull-up over a long execution period (e.g., one million instructions); the bitlines in the "inactive" subarrays are isolated. Due to infrequent switching of precharge devices, the energy overhead of bitline isolation is amortized over aggregated energy savings. Because the "active" subarrays use static pull-up, there is no impact on the latency accessing these subarrays. However, because resizable caches preclude individual subarray precharging control, they cannot exploit bitline isolation and its potential for energy savings to the furthest extent.

Table 2: Base system configuration.

Issue & decode	8 instructions per cycle
Reorder buffer	128 entries
Issue queue	64 entries
Load/Store queue	64 entries
Branch predictor	combination
Register file	128 registers; 16R/8W ports
L1 i-cache	32K; 2-way; 2-cycle; 2RW ports
L1 d-cache	32K; 2-way; 3-cycle; 2RW/2R ports
L2 unified cache	512K; 4-way; 12-cycle latency
Memory	100 cycles + 4 cycles per 8 bytes
MSHRs	8 entries

3 Experimental Setup

In this paper, we evaluate our contributions using a spectrum of CMOS technologies in an aggressive 8-way microprocessor. Our circuit parameters (Table 1) are chosen to represent a wide spectrum of CMOS technologies from recent-past (180nm) to near-future (70nm) technologies. The clock speeds are scaled proportionally to the gate delays and match the aggressive 8 fanout-of-four (FO4) delay for each technology [11]. Therefore, the cycle time stays the same relative to the gate delay and a single pipeline stage employs the same number of logic levels across technologies. We use a modified version of CACTI 3.2 [18] and SPICE for the circuit-level simulations.

Table 2 shows the simulated 8-way 16-stage superscalar processor's base configuration. We measure the access latencies of the major structures, including the register files, issue window, branch predictor and L1/L2 caches, using the modified CACTI tool and adjust the overall pipeline depth accordingly. With the same microarchitecture, the chip dimensions and wire lengths of the simulated processor scales linearly with the technology scaling. Ho, et al. [10] show that innovation in material and aggressive scaling in wire dimensions/spacing make it possible that delays of the wires that scale in length scale with the gate delays for the technologies between 180nm and 50nm. Thus, our assumption of the 8-FO4 clock period ensures that the access penalty (in cycles) of the major structures and the overall pipeline depth remain constant for the technologies studied in this paper. To model deeper pipelines and realistic memory systems, we modified Wattch 1.0 [4].

We examine sixteen applications from the SPEC2000 (*ammp*, *art*, *bzip2*, *equake*, *gcc*, *mcf*, *mesa*, *vortex*, *vpr* and *wupwise*) and Olden (*bh*, *bisort*, *em3d*, *health*, *treeadd*, and *tsp*) benchmark suites. We run entire programs for

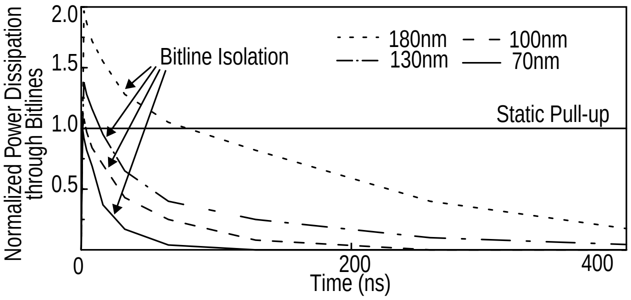


FIGURE 2: Power dissipation through bitlines.

Olden but use SimPoint for SPEC2000 to reduce turnaround time [17]. We gather the subarray pull-up/idle time distributions from the architectural simulations and combine them with the bitline discharge results from the circuit simulations to calculate the overall energy reduction.

4 Bitline Isolation: Energy Overhead & Potential Savings

In this section, we analyze bitline isolation's energy overhead. The energy overhead is a key constraint for designing the subarray selection/identification mechanisms on top of bitline isolation, because careful evaluation of the overhead allows designers to determine: (1) how frequently bitline isolation can be applied, and (2) how aggressive the subarray selection mechanism can be. Assuming an ideal mechanism to identify the accessed subarray (with a perfect hit rate and no identification delay), we evaluate the maximum potential energy savings using bitline isolation.

Energy Implications: Bitline isolation cuts off the leakage paths from the supply voltage to the bitlines by turning off the precharge devices (Figure 1), and reduces the energy dissipated through the paths. Ideally, bitline isolation is expected to completely eliminate the leakage through the bitlines immediately after the precharge devices are turned off. In reality, however, because precharge devices are typically an order of magnitude larger than cell transistors, switching the precharge devices may induce significant current flow and energy dissipation on the bitlines. The current through the bitlines decreases and eventually reaches a steady state, where there may be no additional energy discharge through the bitlines.

Scaling theory [3] predicts that the switching power dissipation in a device reduces by one half with technology scaling, while the leakage power increases by a factor of 3.5. Therefore, the energy overhead in switching the precharge devices is expected to dramatically decrease compared to the static pull-up's bitline discharge for future CMOS technologies.

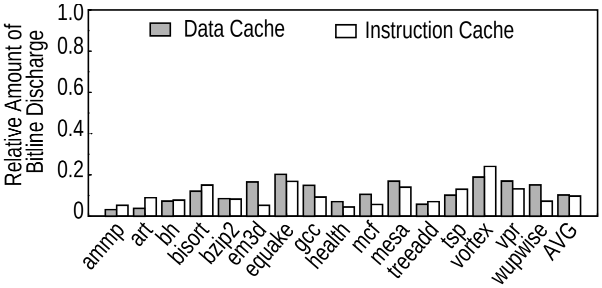


FIGURE 3: Potential bitline discharge savings.

Figure 2 shows the energy overhead trend for various CMOS technologies by presenting the power dissipation through the bitlines in a 1KB subarray as a function of time, after precharge devices are turned off at time zero. For each CMOS technology, the power dissipation is normalized to the one with the static pull-up scheme of its own generation. Because the bitline's steady state voltage level and overall energy dissipation depend on the values stored in the cells connected to the bitline, we assume the worst-case combination of stored values, without affecting the trend of the results.

This figure shows that bitline isolation's energy overhead widely varies across the CMOS technologies. In 180nm technology, the overhead is up to 195% of the power dissipated through statically precharged bitlines. The isolated bitlines reach the steady state over 500ns after they are isolated. However, as expected, the energy overhead and falling time dramatically decrease with technology scaling. Eventually in 70nm technology, very small switching current spike is induced and it melts away quickly, resulting in an insignificant overhead. Therefore, subarray selection mechanisms in the future can be designed to apply bitline isolation more frequently and aggressively for more energy savings.

Potential Energy Savings: From now on, we study an oracle bitline precharging mechanism to quantify the potential energy savings that are achievable by bitline isolation. On every cache access, an oracle identifies the accessed subarray without increasing the access latency and precharges only this subarray. The other subarrays are isolated from the supply voltage to remove unnecessary bitline discharge. Once the access ends, the accessed bitlines are isolated.

If there is little energy overhead in applying bitline isolation, this oracle bitline precharging is most beneficial and ideal to control bitline isolation because it precharges the bitlines only when they need to be precharged. Thus, this oracle study is expected to provide the potential energy savings of bitline isolation in future CMOS technologies where the energy overhead is insignificant.

For the static pull-up scheme, the bitline discharge is constant over time, but the bitline discharge through the isolated bitlines depends on the access interval. If the bitlines are accessed soon after isolation, the energy savings might be insignificant because the bitline discharge remains high while the overhead is consumed. Therefore, although the oracle precharges only one subarray for all the cache accesses, the potential varies depending on the distribution of subarray access intervals.

Figure 3 shows the full potential observed for data and instruction caches in 70nm CMOS technology. In the future the potential bitline discharge reduction is huge: the potential for data and instruction caches are 89% and 90% on average, respectively, each corresponding to 46% and 41% of the overall cache energy saving opportunity.

5 On-Demand Bitline Precharging

In this section, we investigate the timeliness of on-demand bitline precharging. On-demand precharging emulates the oracle bitline precharging studied in Section 4 by partially decoding memory addresses to identify and precharge only the accessed subarrays. Given that bitline isolation's energy overhead in the upcoming generations of CMOS technology is insignificant, and the on-demand subarray identification provides perfect accuracy, the success of on-demand precharging relies purely on the timeliness of its subarray identification mechanism.

In the on-demand precharging scheme with partial address decoding, all bitlines are initially isolated from the supply voltage and approach steady state. On a cache access, part of the address is decoded to identify the relevant subarrays. The isolated bitlines in the relevant subarrays must be pulled up before the decoding is completed and the corresponding wordline is asserted. The delay for partial address decoding and relevant bitline precharging can be hidden if they completely overlap with full address decoding.

To investigate whether these operations can be performed in parallel, we look into the details of the cache's decoder architecture (Figure 4). Without loss of generality, we assume that our decoding logic is similar to that of the CACTI simulator's model [18]. The decoder depicted in Figure 4 contains three major sources of the delay, each corresponding to one of the three decoding stages. In the first stage, the address is fed into the decoders in the subarrays. The second stage in each subarray divides the address into a number of three bit blocks and generates 8-bit one-hot codes via 3-to-8 decoders. NOR gates in the third stage combine these 8-bit one-hot codes to identify the accessed row.

Partial address decoding requires the first and second stages of full address decoding. After the second stage, the outcomes from one or more 3-to-8 decoders are utilized to identify the accessed subarrays, depending on the number of subarrays in the cache. If the cache has eight or fewer subarrays, the relevant subarrays can be identified just before the third stage. Otherwise, the partial decoding needs to combine the outcomes from the second stage using NOR gates with fewer inputs and be further delayed.

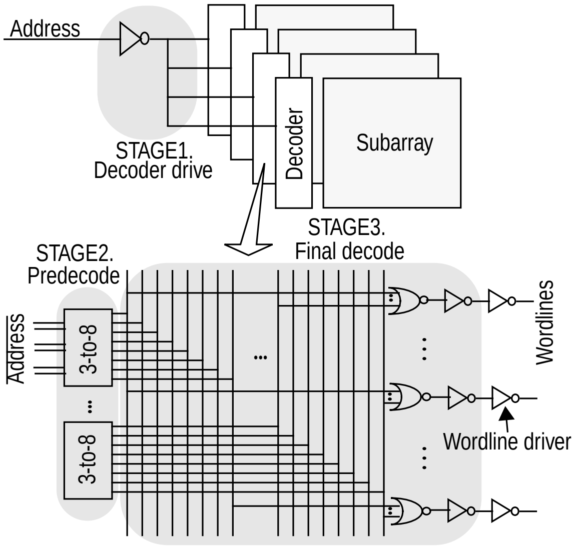


FIGURE 4: Cache address decoder architecture.

The margin of time given to bitline precharging after partial address decoding is slim. With eight or fewer subarrays, the margin of time is the third stage latency of full address decoding. With more subarrays, it is even shorter. Moreover, bitline isolation fully discharges the bitlines in the worst case and pulling up these bitlines can exceed this time margin. In contrast to the fully discharged bitlines, bitline precharging on an active cache access can be overlapped with the address decoding, because active cell reads create only a small voltage drop (0.1 to 0.2V).

The bitline precharging delay depends on the size of the precharge devices and the subarrays. First, the precharging delay decreases as the size of the precharge devices grows. However, we cannot enlarge the precharge devices indefinitely for faster precharging. The static pull-up scheme always turns on the precharge devices and fights against the bitline discharge on an active cell read. Therefore, larger precharge devices slow down the bitline discharge on a cell read and, in turn, increase the cache access latency. In addition, larger precharge devices require more switching energy. Second, the precharging delay decreases as the subarray size decreases, because the bitline length and capacitive load decrease. However, reducing the subarray size increases the number of subarrays in a cache and makes partial address decoding as complicated and slow as full address decoding by requiring more address bits. Moreover, a larger number of subarrays increase the cache area and routing delay.

Table 3: Decode and precharge delay.

Sub-array size	Feature size (nm)	Address decode (ns)			Worst-case bitline pull-up (ns)
		Decode drive	Pre decode	Final decode	
1KB	180	0.25	0.28	0.20	0.39
	130	0.21	0.27	0.16	0.31
	100	0.18	0.21	0.13	0.24
	70	0.12	0.15	0.09	0.16
4KB	180	0.16	0.20	0.18	0.50
	130	0.11	0.15	0.13	0.36
	100	0.088	0.11	0.10	0.28
	70	0.062	0.077	0.07	0.19

Evaluation: Table 3 shows the delays for full address decoding's three stages and bitline precharging for both 1KB and 4KB subarrays and various CMOS technologies. We assume 32-byte cache lines for 32K 2-way set-associative L1 caches. The size of the precharge devices is assumed to be factor of ten larger than the cell transistors. For a 1KB subarray size, the cache has 32 subarrays and partial address decoding is required to combine the outcomes of the second stage to identify the relevant subarrays. For 4KB subarrays, partial address decoding ends immediately after the address decoding's second stage.

Regardless of the subarray size or CMOS technology, we observe that bitline precharging consistently takes longer than the final stage of the address decoding, which is the maximum margin for bitline precharging to overlap the on-demand precharging with full address decoding. This difference delays the cache access latency by one cycle. The average performance impact of this longer cache access latency is 9% and 7% for data and instruction caches, respectively. Therefore, in contrast to the recent proposal [8] that assumes on-demand precharging is applicable without delaying cache accesses, on-demand precharging is not applicable to high-performance cache designs. Instead, successful selective bitline precharging requires *early* subarray identification mechanisms with high accuracy.

6 Gated Precharging

Successful selective precharging in the future must be both timely and accurate. In this section, we propose and analyze *gated precharging* [20], which controls bitline isolation based on the application's subarray reference locality. Gated precharging allows for a timely and accurate subarray identification and achieves *near-optimal* energy savings close to the potential studied in Section 4.

In contrast to oracle or on-demand precharging where the subarrays are disabled immediately after an access, gated precharging leaves the accessed subarray precharged if another access to the subarray is predicted to occur in a short period of time. Therefore, the bitlines are precharged for the next access even before the access begins. In such a way, gated precharging identifies the accessed subarrays early, ensuring timeliness.

Gated precharging does not bound the number of precharged subarrays to one or the associativity of the cache. When the subarray reference locality is low, the technique precharges multiple subarrays in the hope that one of the precharged subarrays is accessed. In the common case of high locality, the technique aggressively precharges only one subarray. This flexibility provides high prediction accuracy.

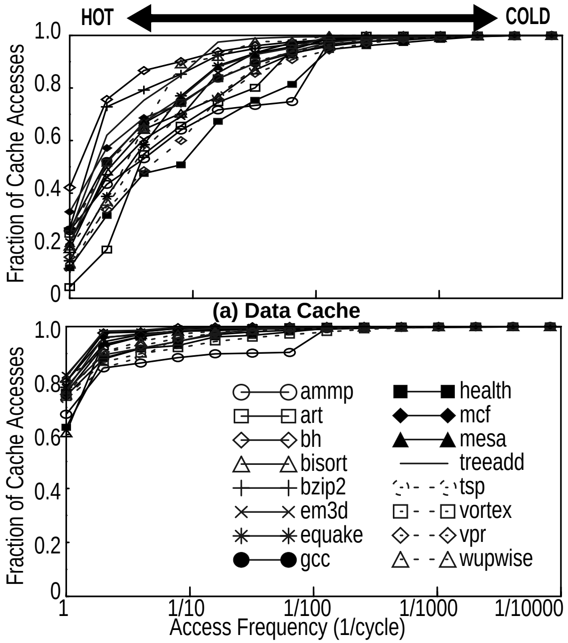
Gated precharging is based on two key observations. First, recently accessed subarrays are most likely to be reused in the short term. We call the subarrays that are currently in frequent use *hot subarrays*. Second, the number of hot subarrays at any moment is typically small. In other words, most cache accesses within a short time are localized to a small number of cache subarrays. The subarray reference locality is inherent to the application's execution. Application programs normally break down computation into distinct program phases. In each phase, a small portion of the application typically iterates and computes over parts of a data structure, resulting in the cache accesses localized within a small number of subarrays.

Gated precharging identifies hot subarrays by exploiting the application's subarray reference locality and applies bitline isolation to the other subarrays. To exploit subarray reference locality, hardware (or software) must provide an accurate mechanism to identify hot subarrays and forecast future subarray usage. In this paper, we use a simple and intuitive hardware mechanism to detect the subarray reference locality and identify hot subarrays.

In the rest of this section, we first study subarray reference locality and show that most cache accesses over a short period are localized to a small number of subarrays. Therefore, recently-used subarrays are most likely to be reused in the near future. Next we describe a simple hardware mechanism to detect and exploit reference locality. We then discuss the hardware/performance overhead of the mechanism and evaluate gated precharging.

6.1 Locality of Cache Subarray References

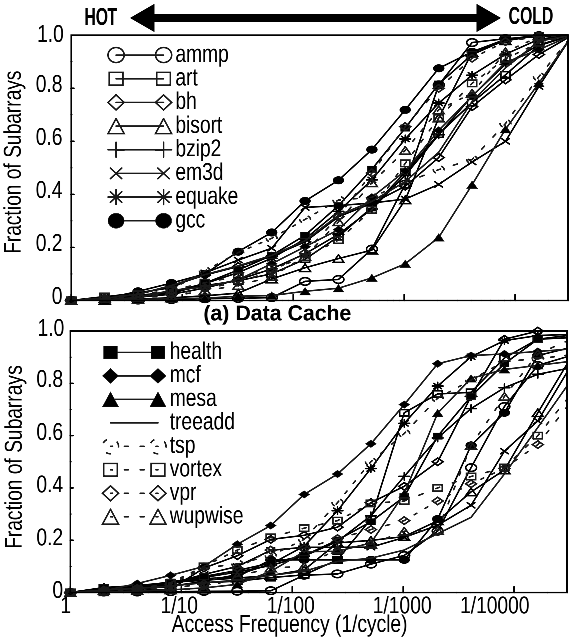
In this section we examine sixteen applications from the SPEC2000 and Olden benchmark suites with a base system configuration described in Section 3 to demonstrate subarray reference locality: most cache accesses



(a) Data Cache

(b) Instruction Cache

FIGURE 5: Cumulative distribution of cache accesses vs. access frequency.



(a) Data Cache

(b) Instruction Cache

FIGURE 6: Fraction of hot subarrays.

occur in a small number of hot subarrays. The hot subarrays vary over the program's dynamic instruction stream.

An important metric in this section is the subarray access frequency (or access interval, where access frequency is the reciprocal of access interval). The access frequency of a subarray indicates how hot the subarray currently is. Therefore, we can think of it as the *temperature* of the subarray: subarrays with a high access frequency (or high temperature) are hot. We will investigate the subarray reference locality as a function of access frequency.

Temporal Locality of Subarray References: Figure 5 shows the cumulative distribution of cache accesses versus the subarray access frequency. This figure indicates how often the accesses occur in hot subarrays. For most of our benchmarks, we observe that a large portion of cache accesses are distributed around a high access frequency (i.e., high temperature). For instance, with the exception of three applications, 95% of data cache accesses occur in subarrays with an access frequency of at least one every 100 cycles, implying that most cache accesses occur in the hot subarrays. Therefore, the hot subarrays are most likely reused, indicating high temporal locality of the subarray accesses. For *ammp*, *art* and *health*, their high cache miss ratios result in a large interval between two accesses and a lower subarray access frequency.

Fraction of Hot Subarrays: Another important observation underlying gated precharging is that the number of hot subarrays is typically small. Figure 6 shows the fraction of hot subarrays in a cache as a function of the access frequency. This figure indicates how many subarrays will be categorized as hot for a given access frequency threshold. A subarray is hot if its access frequency (temperature) is above a certain threshold. The lower the threshold access frequency, the more subarrays will be categorized as hot. An important observation from this figure is that the number of hot subarrays is typically small for a large access frequency threshold. For example, with a threshold of one in 100 cycles, the fraction of hot subarrays in a cache is only 22%, on average. For a 1000-cycle threshold, at most 40% of subarrays are considered hot.

6.2 Implementing Gated Precharging

We have demonstrated that only a small number of subarrays are hot and those hot subarrays have high temporal locality. To exploit this property, gated precharging measures the temporal locality of each subarray and identifies the subarrays with high temporal locality. Gated precharging precharges only hot subarrays, since most accesses are likely to occur in these subarrays in the near future.

Figure 7 depicts an implementation of gated precharging. Gated precharging employs a decay counter per subarray to capture the recent usage of the subarray. The value of the counter is compared to a threshold value every cycle to determine whether the subarray is hot or cold. If the counter value is below the threshold, the subarray was accessed recently and is likely to be reused soon. Therefore, the subarray remains precharged for the next cache

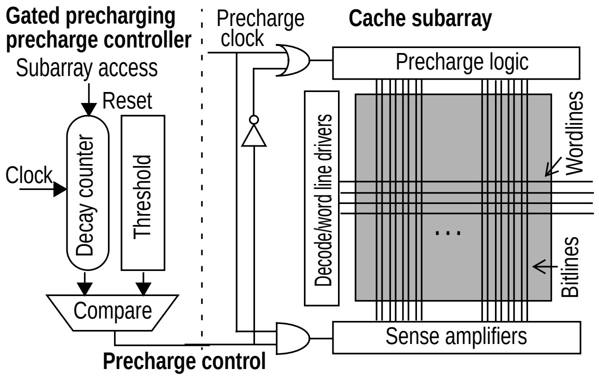


FIGURE 7: A gated precharging implementation.

access. Otherwise, the subarray is not likely to be accessed and its bitlines are isolated.

The key adaptivity parameter of the technique is the threshold value. A small threshold value allows gated precharging to aggressively disable subarrays after a shorter period of inactivity but may result in more mispredictions. Threshold values can be determined in various ways, but studying threshold selection algorithms is beyond the scope of this paper. As a first step to understanding gated precharging, we use both the per-benchmark optimum found from profiling and a constant threshold across the board.

Gated precharging introduces very small and simple additional hardware. As shown in Figure 7, the technique requires only one extra counter and the threshold comparison logic per subarray. Our experiments show that 10-bit decay counters are sufficient. The result of the comparison is fed into the precharge control logic which already exists in conventional caches. Our simulation estimates that the additional hardware structures dissipate less than 0.02% of the energy required for one base cache access.

6.3 Performance Implications

If a cache access occurs on a subarray left isolated by gated precharging, the access is delayed until the corresponding bitlines are precharged. As we have seen in Table 3 (Section 5), the bitline precharging takes one cycle for the spectrum of CMOS generations and clock frequencies. This delay increases the latency of the cache access and degrades performance. For instruction caches, this delay slows the fill-up rate of the instruction fetch queue. As long as the gated precharging's accuracy is high, the performance impact from the delayed fill-ups is expected to be minimal. The performance impact of delayed cache accesses for data caches might be more visible. In highly speculative modern processors, instructions that are dependent on a cache access are speculatively issued assuming that the data from the cache access will be available in a certain cycle. This technique is called load hit speculation [7, 9, 23]. However, uncertainty in the cache latency can cause additional squashes and reissues of the speculatively-issued instructions. Such replays adversely affect the energy dissipation as well as execution time.

Uncertain Load Latency and Instruction Issue: Modern highly speculative superscalar processors including the MIPS R10000, the Alpha 21264 and the Pentium 4, perform load hit speculation [7,9,23]. In general, there is a non-zero delay between instruction issue and execution. Therefore, to ensure back-to-back execution of a load and its dependents (and even their children), instructions depending on a load are speculatively issued as early as possible with the assumption that the load will hit in a cache and the data will be available in a known and fixed number of cycles. However, if the load takes longer or does not provide the data in a given latency, the speculatively-issued instructions must be squashed and reissued.

The major sources of cache access latency variation in conventional processors are L1 cache misses and misspeculated speculative loads (loads issued before preceding store addresses are resolved). As cache misses and load misspeculations are rare, load hit speculation improves performance significantly by executing the load and its dependents back-to-back.

Gated precharging creates another level of uncertainty for cache hit latency, because mispredictions increase cache hit latency. The increased uncertainty of the cache hit latency might incur significant performance degradation in highly speculative modern superscalar processors. The instruction replay affects the performance not only because it delays the execution of dependent instructions, but also because it wastes resource and issue bandwidth that could have been utilized for useful independent instructions.

Conventional processors take two different approaches upon incorrect load hit speculation. Some processors, such as the MIPS R10000 or Alpha 21264, squash and reissue all the instructions following the misspeculated load. Other processors such as the Pentium 4, squash only instructions dependent on the load. The Pentium 4's approach is better at reducing the performance impact but may be more complex. Such an approach is particularly important for long pipelines like in the Pentium 4, because these pipelines exhibit large delays between the load issue and resolution, and there can be a large number of independent instructions issued during this delay. The approach used by the MIPS R10000 might squash all of them, resulting in significant performance and power degradation. As our base system has a 16-stage pipeline and exhibits long load-issue-to-resolution delay (6 cycles), we take the Pentium 4's approach in this study.

Improving Accuracy Using Predecoding: Subarray reference locality in data caches is lower than in instruction caches, and thus we expect gated precharging in data caches to exhibit lower accuracy. Moreover, load hit misspeculation caused by the uncertain load hit latency may amplify the performance impact in data caches. We improve the accuracy of gated precharging in data caches by using a simple heuristic called *predecoding*.

The key observation of predecoding is that, for most of the memory instructions that use displacement addressing (address = base address + displacement), its base address determines the accessed subarray. Displacement addressing is the most commonly used addressing mode in many ISA's. The memory instructions with displacement addressing use a base register and a displacement to determine which address is accessed. Most of the displacement values are small and therefore do not affect which subarray is accessed. Because the accessed subarray can be identified right after the base register is read prior to address calculation, the accessed subarray can be precharged earlier in the pipeline.

For the sixteen applications from SPEC2000 and Olden benchmark suites, we observe that predecoding on 1KB subarrays predicts the accessed subarrays with an 80% of accuracy. Even with subarrays as small as a cache line, an average 61% of the predecoding is accurate. In evaluations, we combine predecoding with gated precharging to achieve a higher accuracy.

6.4 Evaluation

In this section, we present experimental results on the performance and energy of gated precharging. For gated precharging, we present results from the statically-found per-benchmark optimum thresholds with a 1% performance degradation. All the threshold values are on the order of 10 to 1000, with most clustered around 100. As a reference, we show the average savings when a constant threshold (100) is applied to all the benchmarks. The base subarray size is 1KB. We first show the bitline discharge savings achieved through gated precharging (combined with predecoding for data caches.) Then we compare gated precharging against the previously proposed resizable cache technique. Finally, we investigate gated precharging's sensitivity to subarray sizes.

Energy Savings: Figure 8 shows the fraction of precharged subarrays (left bar) and the relative energy dissipation due to bitline discharge (right bar) for the L1 data and instruction caches achieved by gated precharging. The results are normalized with respect to conventional caches of the same configuration. The figure shows that gated precharging significantly reduces the number of subarray prechargings and the amount of the bitline discharge. On average, with a 1% performance degradation, gated precharging precharges only 10% of the subarrays in data caches and 6% in instruction caches, each of which corresponds to approximately three and two subarrays out of 32, respectively. They correspond to 83% and 87% reductions in the bitline discharge. With a constant threshold, the average discharge reductions are 78% and 81%, respectively. The instruction replay (Section 6.3) in the data cache increases the processor's energy consumption by less than 1%. Predecoding increases the data cache's bitline discharge reduction by 6%.

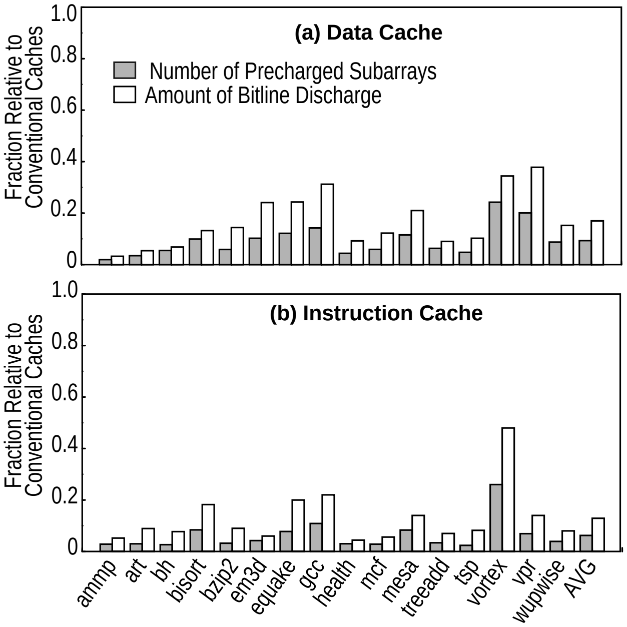


FIGURE 8: Number of precharged subarrays and amount of bitline discharge.

We observe a larger reduction of the bitline discharge in instruction caches than in data caches. Instruction streams have more stable footprints, because they exhibit higher spatial locality at the cache line level. Moreover, the variable load hit latencies in data caches caused by mispredictions produce squashes and reissues of instructions, whereas the delayed load hit in instruction caches merely slows down the instruction fetch queue fill-up. These squashes and reissues have an adverse impact on execution time as well as the cache's energy dissipation. Therefore, data caches require higher subarray identification accuracy than instruction caches to save the same amount of energy with the same performance penalty.

We observe huge reductions in the number of precharged subarrays for applications such as *ammp*, *art* and *health* in data caches. There are two different scenarios that make this possible. For applications like *health*, their small footprint and high subarray reference locality greatly increase the effectiveness of gated precharging.

Gated precharging's capability to capture locality results in a huge reduction in the average number of precharged subarrays. Second, other applications like *ammp* and *art* receive virtually no benefit from having L1 caches. These applications mostly thrash in L1 caches, so the delayed precharging caused by aggressive bitline isolation does not incur a significant performance degradation. Therefore, gated precharging can employ a very aggressive threshold and achieve a large energy savings without a significant performance degradation.

Gated Precharging vs. Resizable Caches: Resizable caches exploit the variability in cache size requirements within and across applications. Resizable caches monitor cache performance at every interval and change the cache size at the granularity of multiple subarrays at the end of each interval. This interval is typically around one million instructions. In this paper, we use the miss ratio as the cache's performance metric and vary both the number of cache sets and set associative ways [22].

Resizable caches switch precharge devices infrequently so the energy overhead of toggling the bitline precharge can be amortized into the large interval. However, the infrequent and coarse-grain characteristics of resizable caches result in suboptimal cache sizes and prevent them from fully utilizing the available potential. Moreover, resizable caches introduce extra cache misses that are created because resizing may require the remapping of data into a cache and because cache downsizing can map two hot subarrays into one. Therefore, resizable caches have a larger performance impact than gated precharging.

Figure 9 compares the relative bitline discharge of gated precharging against that of resizable caches for various CMOS technologies. Each value represents the bitline discharge averaged over the tested benchmarks. The results are obtained as aggressively as possible while maintaining a 1% performance penalty. This figure clearly shows that resizable caches achieve almost a constant bitline discharge regardless of the CMOS technology, whereas gated precharging exhibits a large variation. Resizable caches amortize the overheads into the large switching interval, resulting in a consistent savings across CMOS generations. However, gated precharging switches precharge devices more aggressively, and thus the amount of energy overhead directly impacts the results.

Comparing two techniques in 70nm technology, we observe that many applications in data caches and *equake, gcc, vortex* and *vpr* in instruction caches show a large gap between gated precharging and resizable caches (not shown in the figure). For these applications, conflicts between hot subarrays caused by cache downsizing would produce a large number of cache misses and prevent aggressive downsizing, whereas gated precharging does not incur such conflicts. Therefore, to maintain a performance degradation of less than 1%, resizable caches cannot aggressively downsize the caches beyond certain sizes.

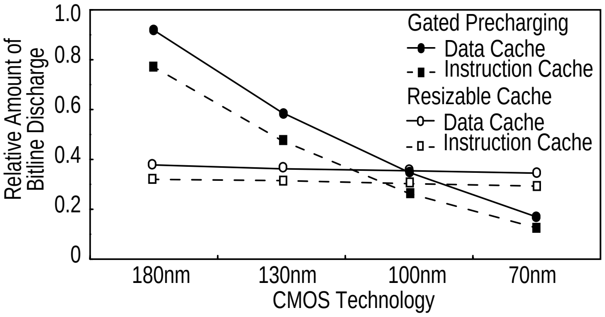


FIGURE 9: Bitline discharges in gated prechargin and resizable caches.

Effect of Subarray Size: Here, we examine how the subarray size affects the effectiveness of gated precharging in 70nm technology and project the future based on these results. The bitline length and subarray size tend to shrink with technology scaling. Two major driving forces for the smaller subarray are the leakage current and the wire delay. With technology scaling, larger leakage through the SRAM cells on the unaccessed rows reduces the voltage differential induced in the bitlines by an active cell read, requiring fewer cells to be attached to the bitlines. Moreover, the relatively longer wire delay in the advanced CMOS technology requires bitlines to be segmented to maintain the cache access latency.

We expect that gated precharging is more effective with smaller subarrays. A large subarray can experience non-uniform access frequencies, and smaller subarrays may capture such non-uniformity effectively to yield finer control on each section of the subarray. However, if the subarray size gets too small, the access frequency for each subarray becomes too small and gated precharging requires larger threshold to reduce the number of delayed cache accesses.

Figure 10 exhibits the fraction of precharged subarrays with the cache subarray sizes of 4KB, 1KB 256B and 64B. A 64B-subarray includes only two cache lines. On average, the relative numbers of precharged subarrays with the subarray sizes from 4KB to 64B are 28%, 10%, 8% and 7% for data caches and 18%, 8%, 6% and 5% for instruction caches. The figure shows that gated precharging works better with smaller subarrays, which suggests that gated precharging will be more effective in the future when caches employ smaller subarray sizes.

We also observe diminishing returns: the effectiveness of gated precharging almost saturates between 64B and 256B. The reasons are two-fold. First, larger subarrays can have a number of sections with different access frequencies within subarrays. Gated precharging for larger subar-

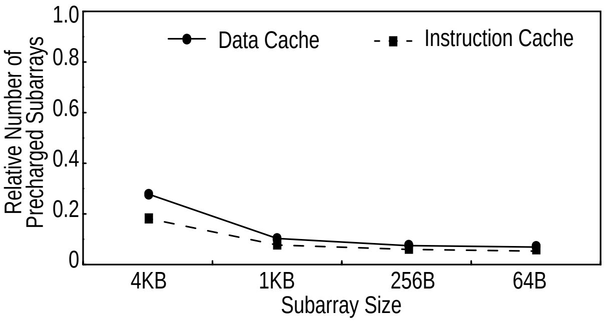


FIGURE 10: Effect of subarray size.

rays controls all the sections' precharging at once, while gated precharging for smaller subarrays controls them separately. Gated precharging for larger subarrays may have a number of prematurely precharged sections within subarrays, and the amount of prematurely precharged sections is more likely to decrease as the subarray size decreases. Second, smaller subarrays require larger thresholds to avoid large performance degradation. However such a conservative threshold setup for smaller subarrays keeps gated precharging from linearly improving as the subarray size decreases.

7 Related Work

A number of previous studies have focused on selective bitline isolation for energy savings in a cache. On-demand (but delayed) precharging was applied to the Alpha 21164 [6] and the StrongArm-110 [14]. However, the large performance overhead of on-demand precharging precludes its applicability to high-performance systems. Resizable caches have been recently proposed by a number of groups [1,16,22]. Yang, et al. studied the key architectural design aspects of resizable caches to evaluate their effectiveness in reducing both cell leakage [21] and bitline discharge through bitline isolation [22]. In this paper, we present results indicating that resizable caches are suboptimal in reducing bitline discharge in future CMOS technologies.

Kim et al. [13] presented a sophisticated, aggressive subarray prediction mechanism to reduce cell leakage in instruction caches. In contrast, we propose techniques for subarray prediction to eliminate bitline discharge (rather than cell leakage) in both instruction and data caches. Moreover, unlike prior proposals for subarray prediction, we carefully analyze the impact of load replay in deep pipelines and consider realistic subarray misprediction latencies and their effect on overall performance.

In addition, several researchers have suggested using way-prediction for energy savings [12,15]. To improve latency, modern set-associative caches overlap tag lookup with data array access resulting in read accesses to all associative ways within a set. Way-predicting caches predict the correct associative way upon a cache access and read data only from the subarrays in the predicted way to reduce energy. In contrast, in this paper we focus on bitline discharge in subarrays that are *not* read upon a cache access. Therefore, way-prediction can be combined orthogonally to our techniques to reduce overall energy.

8 Conclusions

In this paper we carefully quantified the energy and performance trade-offs of bitline isolation and studied its potential savings. Based on these studies, we proposed architectural techniques necessary to realize the full potential of bitline isolation in nanoscale CMOS L1 caches.

We first showed that bitline isolation can be achieved with little energy overhead in near-future CMOS generations, thus aggressive bitline isolation will be a desirable approach to reducing bitline discharge in high-performance nanoscale CMOS caches. We also showed that bitline isolation can potentially achieve 89% (data caches) and 90% (instruction caches) reductions of bitline discharge in 70nm technology.

We proposed and investigated the on-demand precharging technique. However, we showed that the on-demand precharging technique degrades performance significantly because its subarray identification is untimely. To achieve timely and accurate subarray identification, we proposed gated precharging, which exploits subarray reference locality using a simple hardware mechanism. Gated precharging achieves near-optimal bitline precharging by capturing most of the potential. The technique reduces bitline discharge by 83% and 87% and the overall energy dissipation by 42% and 36% for data and instruction caches, respectively, for the SPEC2000 and Olden benchmarks, with only a 1% degradation in performance.

Acknowledgements

We would like to thank Jared Smolens, Roland Wunderlich, the members of Impetus group, and the anonymous reviewers for their helpful comments on earlier drafts of this paper. This work is supported in part by the SRC contracts 2003-HJ-1086 and 2001-HJ-901, the DARPA PAC/C contract F336150214004-AF, an IBM faculty partnership award, and donations from Intel.

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] B. J. Benschneider, A. J. Black, and et. al. A 300-MHz 64-b quad-issue CMOS RISC microprocessor. In *IEEE Journal of Solid-State Circuits*, pages 1203–1214, Nov. 1995.

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

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

[5] A. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of High-Performance Microprocessor Circuits*. IEEE Press, 2001.

[6] 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.

[7] B. Gieseke, et. al. A 600-mhz superscalar risc microprocessor with out-of-order execution. In *ISSCC Digest of Technical Papers*, pages 176–177, Feb. 1997.

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

[9] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the pentium 4 processor. In *Intel Technical Journal*, 2001.

[10] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 39(4):490–504, Apr. 2001.

[11] M. S. Hrishikesh, D. Burger, N. P. Jouppi, S. W. Keckler, K. I. Farkas, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, May 2002.

[12] 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.

[13] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 35)*, pages 219–230, Nov. 2002.

[14] 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.

[15] M. D. Powell, A. Agrawal, T. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via selective direct-mapping and way prediction. In *Proceedings of the 34rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 34)*, Dec. 2001.

[16] 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.

[17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[18] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical Report 2001.2, Compaq Corporation, Western Research Laboratory, Aug. 2001.

[19] S. J. E. Wilton 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.

[20] S.-H. Yang and B. Falsafi. Gated precharging: Using temporal locality of subarrays to save deep-submicron cache energy. In *Proceedings of Workshop on Complexity-Effective Design held in conjunction with the 29th International Symposium on Computer Architecture (ISCA-29)*, May 2002.

[21] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. In *Proceedings of the Seventh IEEE Symposium on High-Performance Computer Architecture*, Jan. 2001.

[22] S.-H. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, pages 151–161, Feb. 2002.

[23] K. C. Yeager. The MIPS R10000 superscalar microprocessor. *IEEE Micro*, 16(2), April 1996.