Journal of Instruction-Level Parallelism 5 (2003) 1-24 Submitted 10/02; published 4/03

Managing Leakage Energy in Cache Hierarchies *

Lin Li LILI@CSE.PSU.EDU
Ismail Kadayif KADAYIF@CSE.PSU.EDU
Yuh-Fang Tsai YTSAI@CSE.PSU.EDU
N. Vijaykrishnan VIJAY@CSE.PSU.EDU
Mahmut Kandemir KANDEMIR@CSE.PSU.EDU
Mary Jane Irwin MJI@CSE.PSU.EDU
Anand Sivasubramaniam ANAND@CSE.PSU.EDU
Microsystems Design Lab, Dept. of Computer Science and Engineering
Pennsylvania State University
220 Pond Lab, University Park, PA 16802 USA

Abstract

Energy management is important for a spectrum of systems ranging from high-performance architectures to low-end mobile and embedded devices. With the increasing number of transistors, smaller feature sizes, lower supply and threshold voltages, the focus on energy optimization is shifting from dynamic to leakage energy. In fact, leakage energy is projected to become the dominant portion of the chip power budget for 0.10 micron technology and below. Leakage energy is of particular concern in dense cache memories that form a major portion of the transistor budget. In this work, we present several architectural techniques that exploit the data duplication across the different levels of cache hierarchy. Specifically, we employ both state-preserving (data-retaining) and state-destroying leakage control mechanisms to L2 subblocks when their data also exist in L1. Using a set of MediaBench and SPEC CINT2000 benchmarks, we demonstrate the effectiveness of the proposed techniques through cycle-accurate simulation. We also compare our schemes with the previously proposed cache decay policy. This comparison indicates that one of our schemes generates competitive results with cache decay. Furthermore, we show how both techniques can be applied in conjunction to provide additional energy gains.

1. Introduction

Leakage power of the chip is expected to increase by five times for each technology generation in the future. This trend will result in leakage power becoming the dominant part of the chip power budget for 0.10 micron technology and below [1]. While dynamic energy will still remain a concern for components that are exercised and switched often, leakage energy is of particular concern in the bulky memory structures. This is due to three reasons: increasing sub-threshold leakage current, leakage energy increases with the effective number of transistors in the circuit, and a large transistor budget is allocated for on-chip memories in current processors.

Many techniques have been proposed in the past to reduce cache energy consumption. Among these are partitioning large caches into smaller structures to reduce the dynamic energy [2, 3] and the use of a memory hierarchy that attempts to capture most accesses in the smallest size memory. By accessing the tag and data array in series, Alpha 21164's L2 cache [1] can access the selected

*. This work was supported in part by grants from GSRC, NSF Grants 0103583, 0082064 and CAREER Awards 0093082, 0093085.

cache bank for energy efficiency. In [4, 5], way-prediction is used to reduce energy consumption of set-associative caches. Selective cache ways [6] varies the number of ways for different application requirements. In [7], a small filter cache is placed prior to L1 cache to reduce energy consumption. Dynamic Zero Compression [8] employs single-bit access for zero-valued byte in the cache to reduce energy consumption. However, most of these techniques do little to alleviate the leakage energy problem as the memory cells in all partitions and all levels of the hierarchy continue to consume leakage power as long as the power supply is maintained to them, irrespective of whether they are used or not. Various circuit technologies have been designed specifically to reduce leakage power when the component is not in use. Some of these techniques focus on reducing leakage during idle cycles of the component by turning off the supply voltage. One such scheme, gated-Vdd, was integrated into the architecture of caches [9] to dynamically shutdown portions of the cache. This technique was applied at a cache block granularity in [10] and used in conjunction with software to remove dead objects in [11]. However, all these techniques assume that the state (contents) of the supply-gated cache memory is lost. While totally eliminating the supply voltage results in the state of the cache memory being lost, it is possible to apply a state-preserving leakage optimization technique if a small supply voltage is maintained to the memory cell. There are many alternate implementations that have been recently proposed at the circuit level to achieve such a state-preserving leakage control mechanism [12, 13, 14]. As an abstraction of these techniques, the choice between the state-preserving and state-destroying techniques depends on the relative overhead of the additional leakage required to maintain the state as opposed to the cost of restoring the lost state from other levels of the memory hierarchy.

An important requirement to reduce leakage energy using either a state-preserving or a state-destroying leakage control mechanism is the ability to identify unused resources (or data contained in them). In [9], the cache size is reduced (or increased) dynamically to optimize the utility of the cache. In [10], the cache block is supply-gated if it has not been accessed for a period of time. In [15], hardware tracks the hypothetical miss rate and the real miss rate by keeping tag line active when deactiving a cache line. And then the turn-off interval can be dynamically adjusted based on such information. In [16, 17], dynamic supply voltage scaling is used to reduce the leakage in the unused portions of the memory. In contrast to the other schemes, it also preserves data when in low leakage mode. The usefulness and practicality of such state-preserving voltage scaling schemes for embedded power-optimized memories is demonstrated in [18]. The focus in [19] is on reducing bitline leakage power using leakage-biased bitlines. The technique turns off precharging transistors of unused subbanks to reduce bitline leakage, and actual bitline precharging is delayed until the subbank is accessed. In comparison to the prior efforts at optimizing memory leakage, in this work, we focus on exploiting the data duplication present in an on-chip L1-L2 cache hierarchy (which consists of an L1 instruction cache, an L1 data cache, and a unified L2 cache) to apply the leakage control mechanisms. For example, in general, in an L1-L2 cache hierarchy, the data present in L1 is also contained in L2, when destructive interference in L2 does not happen between instruction and data. Our goal is to transition the cache subblock in L2 to a standby leakage mode when its data is moved to L1. The goal is to save leakage energy by keeping only one active copy of the data. Since the subblocks in L2 moved to L1 are the ones that are most recently used, the cache decay mechanisms proposed previously do not immediately target these subblocks for leakage optimization. Thus, the mechanism proposed in this paper can be applied in conjunction with other existing leakage control mechanisms. Two-level exclusive cache schemes have also been proposed for improving performance [20]. Our technique mimics exclusion by putting a duplicated copy to sleep mode.

In this paper, we make the following contributions:

- Based on state-preserving and state-destroying leakage mechanisms, we describe five leakage reduction strategies that exploit data duplication in the cache hierarchy. Using a set of MediaBench and SPEC CINT2000 benchmarks, we demonstrate the effectiveness of the proposed techniques through cycle-accurate simulation.
- We compare our schemes with cache decay policy, a finite state machine (FSM) based strategy that turns off cache subblocks when they are idle for a sufficiently long period of time. This comparison indicates that one of our schemes is competitive in terms of energy savings. Furthermore, we show how both techniques can be applied in conjunction to provide additional energy gains.

The remainder of this paper is organized as follows. In the next section, we present technology and circuit support for leakage energy optimization. In Section 3, we present five leakage energy optimization strategies in detail. We compare and integrate these strategies with cache decay in Section 4. In Section 5, we measure the sensitivity of leakage optimization techniques to different parameters. Finally, in Section 6, we summarize our major contributions and give an outline of planned future research on this topic.

2. Technology/Circuit Support for Leakage Control

As feature sizes of transistors continue to decrease, the supply voltage has to be scaled to keep current densities in check providing quadratic dynamic energy savings. However, the increasing number of transistors and increased clock frequencies have aggravated the power consumption problem. Another significant technology trend has been the corresponding decrease in threshold voltage, V_t, along with the supply voltage. This trend is due to the fact that the switching speed of a transistor is a function of the difference between the supply voltage and threshold voltage. From the leakage energy perspective, this trend is detrimental as leakage energy increases exponentially with a decrease in threshold voltage.

Many circuit techniques have thus focused on limiting the device leakage. One such technique is the use of high-V_t transistors on non-critical paths of the circuit [1]. This technique can help to reduce leakage in these paths when they are used or idle. Another technique that can be employed is to introduce a power-switch between the power supply and the leaking circuit. The PMOS switch provides a virtual supply voltage to the leaking circuit. The switch is turned off when the circuit is idle to cut off the supply voltage, eliminating leakage energy. The third technique is dynamic scaling of the supply voltages of the circuit. Due to short-channel effects, the leakage current reduces significantly when supply voltage reduces.

We select the dynamic scaling of the supply voltages as our leakage reduction technique. Whether the data in memory cell is retained or not depends on the choice of the supply voltage. In normal mode, the supply voltage of memory cell is 1.0V. In low leakage control mode, we use 0.3V supply voltage for state-preserving mechanism and 0V supply voltage for state-destroying mechanism.

There are additional constraints imposed by the state-preserving leakage control. For the state-preserving leakage control, it is also important to prevent memory cells that maintain their state using a small voltage from losing state when connected with the bit lines. This can be avoided

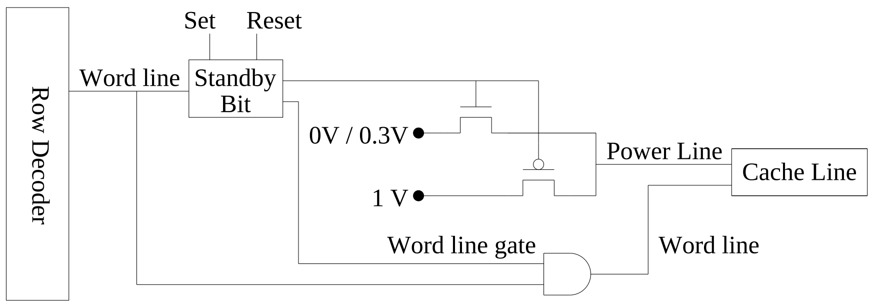

Figure 1: **Leakage control circuitry. Note that the 0V or 0.3V determined at the design time based on whether we require state-destroying or state-preserving mode.**

by ensuring that the wordline signal that controls the connection between the memory cell and the bitlines is suppressed until the memory cells of a cache line in state-preserving leakage control mode recover to a normal supply voltage. This logic can easily be incorporated in the row decoders. A final consideration in the state-preserving mode is that the circuit operating at a lower supply voltage is more susceptible to bit flipping due to soft errors from alpha particle strikes [21]. These soft errors for memories are presently addressed using additional error correction bits. However, as the possibility of such errors increases with reduction in voltage, a more detailed analysis is required to accurately account for its effect. There are interesting tradeoffs offered by the amount of leakage energy that can be saved and the internal node voltage levels (that in turn influence susceptibility to errors). These tradeoffs are planned as a part of our future work.

We use a similar circuit to that proposed in [16] except for the change in supply voltage for state-preserving and state-destroying. In our implementations, for each cache line, a standby bit is added to control the selection of voltage supply of the cell as shown in Figure 1. For both state-preserving and state-destroying, we turn on the standby bit of cache blocks during low leakage control mode and turn it off when the cache subblock needs to be accessed and return to the normal mode. The logic for setting and resetting the standby bit can be incorporated in the cache refill logic of the controller. The conditions for setting and resetting are dependent on the optimization strategy and are explained in the next section. In this work, we assume 10% of original leakage in the state-preserving mode. For the state-destroying mode, on the other hand, we assume no leakage energy consumption. These assumption are based on trends obtained from HSPICE circuit simulation (using Berkeley predictive model [22]). We used a 1.0V, 0.07 micron technology with a normal V_{Tn}/V_{Tp} of 200mV/-220mV and a high V_{Tn}/V_{Tp} of 400mV/-420mV. Simulation were done using a temperature of 85 degrees C. There is an additional area penalty associated with the leakage control circuitry. Our estimates indicate that the power switch, word line gating logic, and the standby control increase the area of the cache by 3%. This can potentially increase the length of the wires and have a small impact on the dynamic energy. We also experiment by varying the leakage energy saving parameter in Section 5 to accommodate anticipated variations due to operating condition and circuit parameter variations.

There is additional dynamic energy consumed in switching these transistors that is reflected as *control energy* (also called *control overhead*) in our experiments. Further, cache line in low leakage control mode must be switched to normal mode (voltage settling to normal 1V) before access is permitted. This can be achieved by using the wordline trigger to reset the standby bit in 1 clock cycle. The voltage settling time of 1 cycle is a key difference from the circuit we used in [23], that had a larger penalty. The switching times for both the circuits were validated through HSPICE circuit simulation.

If a cache block is not accessed after being placed into low-power mode (using a state-preserving or state-destroying strategy), we can expect that the state-destroying mode would save more leakage energy than the state-preserving mode as the latter still consumes 10% of the original leakage energy in the standby state. (A more detailed evaluation of relationship between supply voltage scaling and leakage reduction in our employed circuit can be obtained from [24].) However, if, after being put into the low-power mode, the cache block is accessed (either due to the reference residing there or due to some other reference), the state-destroying mode pays a high performance and energy penalty (as the data needs to be accessed from memory). In contrast, under the same scenario, a cache block in the state-preserving state only needs to be reactivated. Therefore, whether state-preserving mode performs better than state-destroying mode depends largely on the duration of idleness for the cache block in question. This is, obviously, a characteristic of application access pattern and cache hierarchy configuration. In our experiments, all cache lines are in the leakage-control mode before their first use for all strategies.

3. Leakage Optimization Strategies and Results

In this section, we present a set of strategies that exploit the state-preserving and state-destroying leakage energy optimization mechanisms and present energy and energy-delay numbers. As explained below, these strategies differ from each other with respect to the circuit type that they employ (state-destroying versus state-preserving), whether they conservatively or speculatively turn off L2 subblocks, and the time that the L2 subblocks are reactivated (powered-on). All strategies power-manage portions of L2 blocks at the subblock granularity. A subblock of L2 is the same size as a block of L1 and is the unit of transfer between L1 and L2.

3.1 Optimization Strategies

- `Conservative:` In this strategy, when a block in L1 is written to, the corresponding subblock in L2 is turned off by setting the standby bit, thereby destroying data and saving leakage in L2. This is a conservative strategy as, before turning off the subblock in L2, it waits until the corresponding block in L1 becomes dirty. Note that this strategy deactivates only dead L2 blocks (as they are written in L1) and this characteristic makes it different from the remaining strategies considered in this paper. It should also be noted that this strategy cannot optimize instruction accesses as instructions are not written.
- `S-SP-Lazy (Speculative, State-Preserving, and Lazy):` In this strategy, when data is brought from L2 to L1, the corresponding L2 subblock is put in a state-preserving leakage control mode. Consequently, as compared to the conservative strategy described above, this strategy has two important differences: it does not wait for the cache block in L1 to become dirty (i.e., it speculatively turns off the L2 subblock) and it does not lose data in L2. If the block in L1 is evicted, no action is performed if the L1 block is not dirty and the corresponding L2 subblock remains in state-preserving leakage control mode. Therefore, number of L2 subblocks in sleep mode can be larger than the number of L1 blocks. However, as in other strategies, if the evicted L1 block is dirty, the corresponding L2 subblock

Strategy	When is L2 subblock turned off?	Energy-saving mechanism in L2	When is L2 subblock reactivated?
`Conservative`	when L1 block becomes dirty	state-destroying	when accessed
`S-SP-Lazy`	when L2 subblock is moved to L1	state-preserving	when accessed
`S-SD-Lazy`	when L2 subblock is moved to L1	state-destroying	when accessed
`S-SP-Immed`	when L2 subblock is moved to L1	state-preserving	when L1 block is evicted
`S-SD-Immed`	when L2 subblock is moved to L1	state-destroying	when L1 block is evicted

Table 1: **Proposed leakage energy saving strategies.**

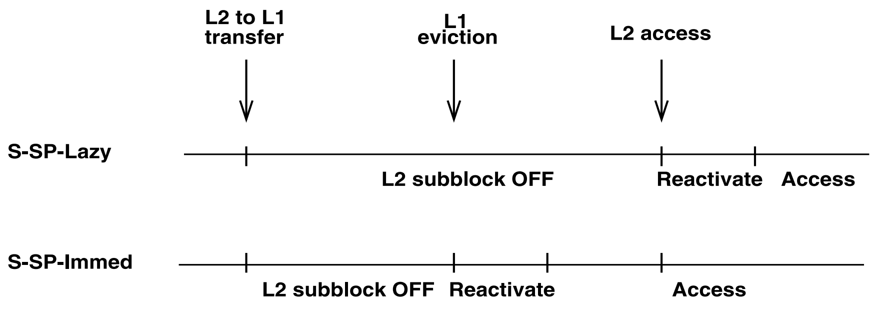

Figure 2: **Comparison of S-SP-Lazy and S-SP-Immed.**

is reactivated and written into. Since a write buffer is employed, the performance penalty of the reactivation period can usually be masked.

- `S-SD-Lazy (Speculative, State-Destroying, and Lazy):` This strategy is similar to S-SP-Lazy, the difference being that the subblock in L2 is put in the state-destroying mode. So, as long as the subblock in L2 is in the powered off state, this strategy saves more leakage energy than S-SP-Lazy (which uses the state-preserving mode). On the other hand, when the L2 subblock needs to be accessed, this strategy pays a higher price than S-SP-Lazy as it needs to access the off-chip memory (as opposed to S-SP-Lazy which simply reactivates the L2 subblock).
- `S-SP-Immed (Speculative, State-Preserving, and Immediate):` This is also similar to S-SP-Lazy except that the L2 subblock is reactivated whenever the corresponding L1 cache block needs to be replaced. This early reactivation (as compared to S-SP-Lazy where reactivation occurs only when the L2 block is accessed) can reduce energy savings compared to S-SP-Lazy. However, it has better performance behavior, as when the L2 cache block is accessed, a separate reactivation time is not spent. This situation is depicted in Figure 2. In the S-SP-Lazy case, the cache subblock is in the state-preserving leakage control mode between the time it is moved to L1 and the time that the L2 is accessed, whereas in S-SP-Immed, it is reactivated when the L1 eviction occurs. Consequently, in S-SP-Immed, the L2 subblock reactivation time can be hidden.
- `S-SD-Immed (Speculative, State-Destroying, and Immediate):` This strategy is similar to S-SD-Lazy except that the L2 subblock is reactivated and written back whenever the corresponding L1 cache block needs to be replaced. Its relative merits with respect to S-SD-Lazy are similar to those of S-SP-Immed with respect to S-SP-Lazy. Similarly, its advantages/disadvantages compared to S-SP-Immed are similar to those of S-SD-Lazy compared to S-SP-Lazy.

Simulation Parameter	**Value**
Processor Core	
Functional Units	4 integer and 4 FP ALUs 1 integer multiplier/divider 1 FP multiplier/divider
LSQ Size	32 Instructions
RUU Size	64 Instructions
Fetch Width	4 instructions/cycle
Decode Width	4 instructions/cycle
Issue Width	4 instructions/cycle
Commit Width	4 instructions/cycle
Fetch Queue Size	4 instructions
Cycle Time	0.5ns
Cache & Memory Hierarchy	
L1 Instruction Cache	32KB, 32 byte blocks, 2-way, 1 cycle latency
L1 Data Cache	32KB, 32 byte blocks, 2-way, 1 cycle latency
L2 Cache	1MB unifi ed, 2-way, 128 byte blocks, 10 cycle latency
Data TLB	128 entries, full-associative, 30 cycle miss latency
Instruction TLB	64 entries, full-associative, 30 cycle miss latency
Memory	100 cycle latency
Energy Management	
Technology	0.07 micron
Supply Voltage	1.0V
Voltage Supply Settling Time	1 cycle
Dynamic Energy per L1 Access	0.565nJ
Dynamic Energy per L2 Access	5.83nJ
Leakage Energy per L1 Block per Active Cycle	0.551pJ
Leakage Energy per L2 Subblock per Standby Cycle (state-preserving)	0.055pJ
Leakage Energy per L2 Subblock per Standby Cycle (state-destroying)	0pJ
Control Energy	0.055nJ

Table 2: **Our base configuration.**

Table 1 summarizes these four strategies highlighting their differences. It should be noted, however, that when an evicted L1 cache block is dirty, the corresponding L2 subblock needs to be reactivated (by resetting the standby bit) irrespective of the energy-saving strategy used. Therefore, this case is not listed separately under the last column in Table 1. It also needs to be mentioned that in state-destroying modes with set-associative caches, when all subblocks in a given L2 block are moved to L1, this L2 block is invalidated, becoming a suitable candidate for the next cache block replacement in L2. However, if there is a single valid subblock in the block, the block is considered valid and participates in the LRU replacement process.

Benchmark	Input	Execution Cycles (millions)	Cache Energy Leakage (mJ)	Cache Energy Dynamic (mJ)	L2 Leakage
`adpcm-rawcaudio`	clinton.pcm	4.74	1.88 (17.46%)	8.90 (82.54%)	57.28%
`adpcm-rawdaudio`	clinton.adpcm	4.00	1.54 (18.37%)	6.85 (81.63%)	56.41%
`cjpeg`	testimg.ppm	7.69	54.25 (72.97%)	20.09 (27.03%)	77.76%
`djpeg`	testimg.jpg	2.93	13.61 (71.22%)	5.49 (28.78%)	68.77%
`epic`	test_image.pgm	21.33	352.65 (85.62%)	59.20 (14.38%)	90.53%
`unepic`	test.image.pgm.E	5.53	75.31 (88.29%)	9.98 (11.71%)	88.88%
`g721-decode`	clinton.g721	119.09	338.40 (50.47%)	332.10 (49.53%)	55.14%
`g721-encode`	clinton.pcm	123.98	353.54 (50.85%)	341.75 (49.15%)	55.30%
`mesa-mipmap`	-	37.09	1032.67 (92.75%)	80.75 (7.25%)	94.02%
`mesa-osdemo`	-	11.97	315.95 (91.48%)	29.40 (8.52%)	93.77%
`mpeg2-decode`	mei16v2.m2v	65.36	638.12 (75.62%)	205.71 (24.38%)	84.30%
`gzip`	input.source	105.19	2299.63 (89.26%)	276.60 (10.74%)	93.73%
`vpr`	net.in arch.in place.in	160.84	4248.19 (93.62%)	289.69 (6.38%)	93.89%
`gcc`	scilab.i	220.77	5464.22 (93.84%)	358.96 (6.16%)	93.24%
`mcf`	inp.in	171.50	4861.76 (93.68%)	327.75 (6.32%)	94.80%
`perlbmk`	2.1.dict -batch ref.in	133.28	3507.31 (92.68%)	276.95 (7.32%)	93.98%
`vortex`	bendian1.raw	325.33	7760.05 (95.39%)	375.26 (4.61%)	92.97%
`bzip2`	input.source	779.39	21882.9 (98.65%)	299.60 (1.35%)	95.46%
`twolf`	ref	448.16	12462.3 (97.13%)	367.63 (2.87%)	94.50%

Table 3: **Benchmarks used in our experiments and their important characteristics.**

3.2 Simulation Parameters and Benchmarks

We used Simplescalar 3.0 [25] to implement our energy-saving optimization strategies. Simplescalar is a tool-set to simulate application programs on a range of modern processors and systems using fast execution-driven simulation. It provides a detailed simulator for an out-of-order issue processor that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In this work, we used the *sim-outorder* component. Table 2 gives the simulation parameters used for our base configuration.

We used the 70nm technology and energy models from CACTI 3.0 to get the dynamic energies of accessing L1 and L2 caches. We assume that the leakage energy per cycle of the entire L1 cache is equal to the dynamic energy consumed per access. Further, we assume that the leakage of the L2 subblock is equal to that of the L1 block. We evaluated the effectiveness of these strategies using a set of benchmark programs. Our benchmarks are codes from MediaBench suite [26] and SPEC CINT2000 [27] benchmarks. We selected these two groups of codes as they represent different access patterns. For each code in MediaBench suite, the simulations are run to completion. Except bzip2 and mcf, benchmarks in SPEC CINT2000 suits are first fast forwarded 300 million instructions and then simulated 200 million instructions. No instruction is fast forwarded for bzip2 and mcf due to their specific characteristics [28]. The important characteristics of these benchmarks are listed in Table 3. The fourth and fifth columns in this figure give the total leakage and dynamic energy consumptions, respectively, in L1-L2 cache hierarchy assuming all blocks are powered off until their first use and never turned off after that. The last column, on the other hand, gives the percentage contribution of the L2 cache to overall leakage energy consumption in the cache hierarchy. It can be observed that these codes expend a large percentage of leakage energy (77.3% of the cache hierarchy energy on the average) and expend a large fraction of this leakage energy (82.9% on the average [1]) in L2 due to its much larger capacity. Consequently, we can expect large leakage energy savings using our strategies. Note that percentage of dynamic energy is dependent on the

1. This is when no leakage optimization is applied

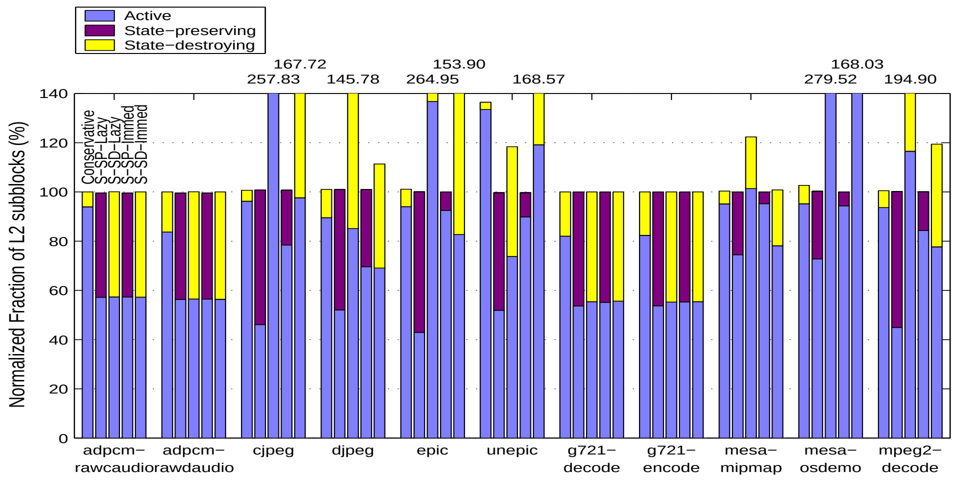

Figure 3: **Normalized fraction of subblocks in L2 in different states. (MediaBench)**

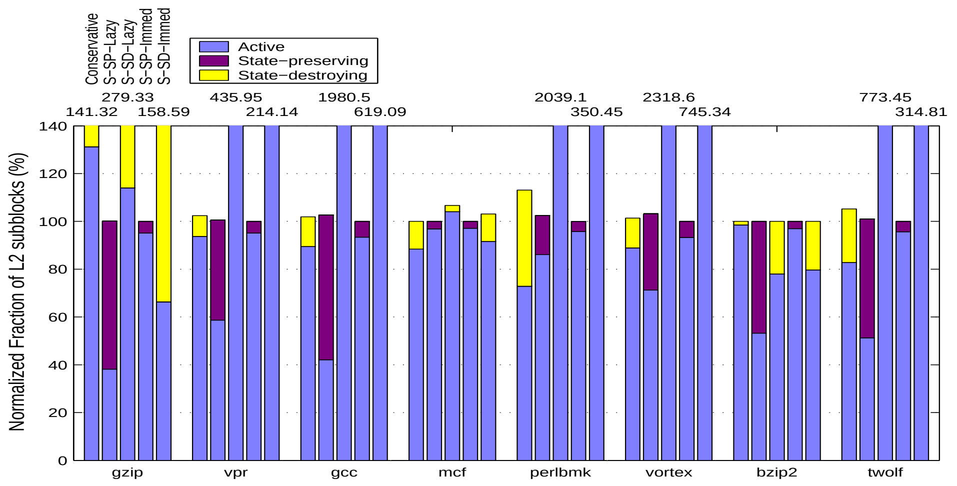

Figure 4: **Normalized fraction of subblocks in L2 in different states. (SPEC CINT2000)**

size of the memory and the number and duration between memory accesses. Most of the energy results given in following subsections are results *normalized* with respect to the values in the fourth and fifth columns of Table 3.

3.3 Impact of Our Strategies

Figure 3 and Figure 4 show the fraction of subblocks in L2 in active, state-destroying, and state-preserving states. Each portion of the bar is the sum of the number of L2 subblocks in active, state-preserving, or state-destroying states (excluding those that were never used in the entire execution) over each clock cycle normalized to the sum of active L2 subblocks over each clock cycle in the original execution. In Conservative, S-SD-Lazy, and S-SD-Immed, subblocks in L2 are distributed between active and state-destroying states. In S-SP-Lazy and S-SP-Immed, subblocks in L2 are distributed between active and state-preserving states. The reason that some bars exceed 100%

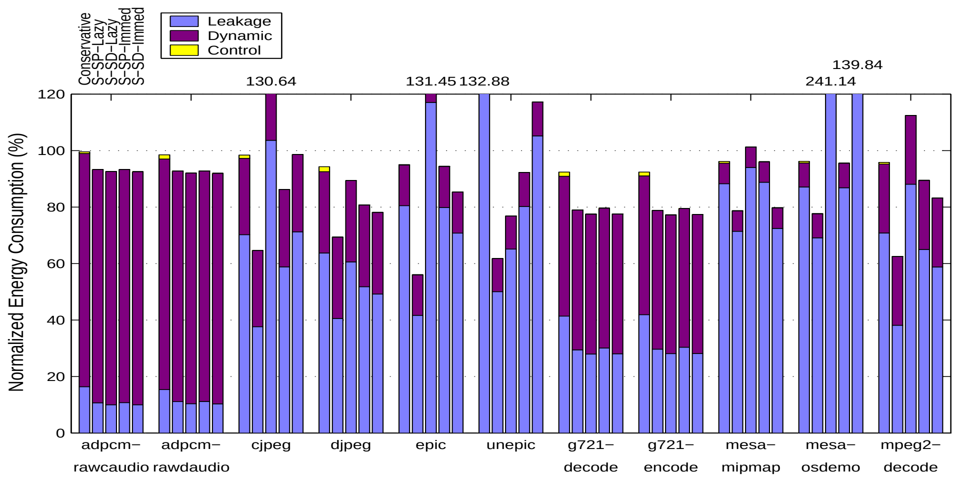

Figure 5: **Normalized energy consumption of our optimization strategies. (MediaBench)**

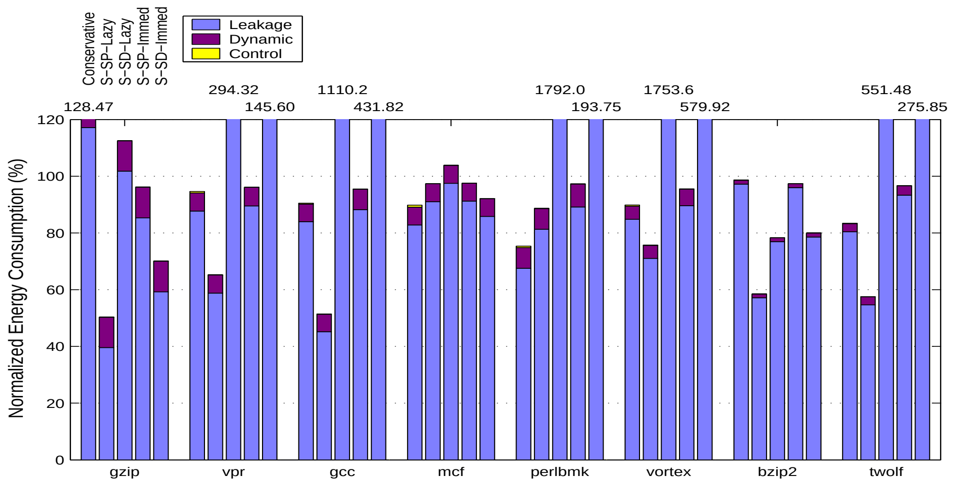

Figure 6: **Normalized energy consumption of our optimization strategies. (SPEC CINT2000)**

is due to the fact that some schemes impact performance and the execution times of benchmarks in such schemes are longer than those in the original execution. Therefore, the total number of subblocks over each clock cycle is larger than 100% after normalization. We can observe that a large fraction of subblocks is in state-preserving and state-destroying states for most benchmarks.

Figure 5 and Figure 6 show the normalized energy consumption for our five optimization strategies. Each bar in this figure is divided into three parts: leakage energy, dynamic energy, and control overhead (energy) and is normalized to the total energy consumed by original execution of benchmarks, which is the sum of the fourth and fifth columns of Table 3. We report dynamic energy because our leakage optimization strategies may increase dynamic energy consumption. We can make the following observations from these results. First, the control overhead is nearly negligible. This is because, compared to the total number of cache accesses, the number of state transitions is very low. In fact, we observe control energy overhead only in the Conservative strategy. This makes sense as in this strategy the control overhead is directly proportional to the number of L1 writes. Second, among our five strategies, S-SP-Lazy generates the best energy results. It reduces leakage energy consumption by 37.7% on the average across all benchmarks and overall cache energy (including dynamic energy and control overhead as well) by 28.5% on the average. On the other hand, the average leakage (the average overall energy) improvements due to Conservative, S-SD-Lazy, S-SP-Immed, and S-SD-Immed are 5.9% (3.0%), -281.0% (-269.4%), 14.6% (7.7%), and -48.1% (-52.1%), respectively. (A negative value indicates an increase in energy consumption).

We now explain why these different strategies performed the way they did. Comparing S-SP-Lazy and S-SD-Lazy, recall that neither of them reactivates L2 subblock when the corresponding L1 block is evicted. If a clean block in L1 is evicted, the corresponding subblock in L2 is in the state-preserving state for S-SP-Lazy but is in the state-destroying state for S-SD-Lazy. Then, when the next access occurs, S-SP-Lazy will incur 1 cycle delay (for reactivation), whereas S-SD-Lazy will lead to 100 cycle penalty (for memory access). During this long memory access all active L1 and L2 blocks leak. Consequently, in most of our codes, S-SP-Lazy exhibits a better energy behavior than S-SD-Lazy. There are, however, exceptions to this general trend: `adpcm-rawcaudio`, `adpcm-rawdaudio`, `g721-decode`, and `g721-encode`. In these codes, the L1 replacement rate (the ratio between the number of L1 replacements and total L1 accesses) is very low (around 0.0003%) and L2 subblocks stay in energy-saving state longer (which works in favor of S-SD-Lazy).

We now focus on S-SP-Immed and S-SD-Immed. Recall that these two schemes differ from S-SP-Lazy and S-SD-Lazy in that they reactivate the L2 subblock when the corresponding L1 subblock is evicted. When data is moved from L2 to L1, S-SP-Immed places the corresponding subblock into the state-preserving state, whereas S-SD-Immed puts it in the state-destroying mode. So, as far as a single subblock is concerned, S-SD-Immed seems to be more energy-efficient. However, if all subblocks in a given L2 block are moved into L1, S-SD-Immed invalidates the entire L2 cache block (i.e., makes it available for replacement). After that, when a new access is made to this cache block, a miss is incurred and main memory needs to be accessed (a 100 cycle delay). During this memory access all active cache blocks in L1 and L2 consume leakage energy. Although the early reactivation (i.e., reactivation in L1 block eviction time) tries to write data back from the L1 cache to the L2 cache, this operation succeeds only when the cache block is in the valid state (i.e., there exists at least a single valid L2 subblock in the cache block). In our scenario, the early reactivation succeeds in S-SP-Immed but fails in S-SD-Immed. Consequently, in such cases, S-SP-Immed might perform better than S-SD-Immed. The results in Figure 5 and Figure 6 indicate that in eight benchmarks S-SP-Immed consumes less energy than S-SD-Immed (due to frequent memory accesses). In the remaining codes, S-SD-Immed performs better than S-SP-Immed (due to lack of the above mentioned scenario).

When we compare S-SP-Lazy and S-SP-Immed, we see that both of them preserve the data in L2, but S-SP-Immed reactivates the subblock in L2 when the corresponding block is evicted from L1. Therefore, it tends to maintain the same execution time as the original (unoptimized) case, incurring some extra energy due to early reactivation. Therefore, its energy behavior is worse than S-SP-Lazy. However, its performance is better than S-SP-Lazy in almost all cases. Finally, comparing S-SD-Lazy and S-SD-Immed, we observe that although both of them destroy data in L2, S-SD-Immed has a better chance for avoiding main memory access, thanks to the early reactivation. In most of the benchmarks, S-SD-Immed consumes less energy than S-SD-Lazy.

Energy consumed in the cache system is only a part of this picture. To perform a fair comparison between the different energy optimization strategies, we also need to account for the additional

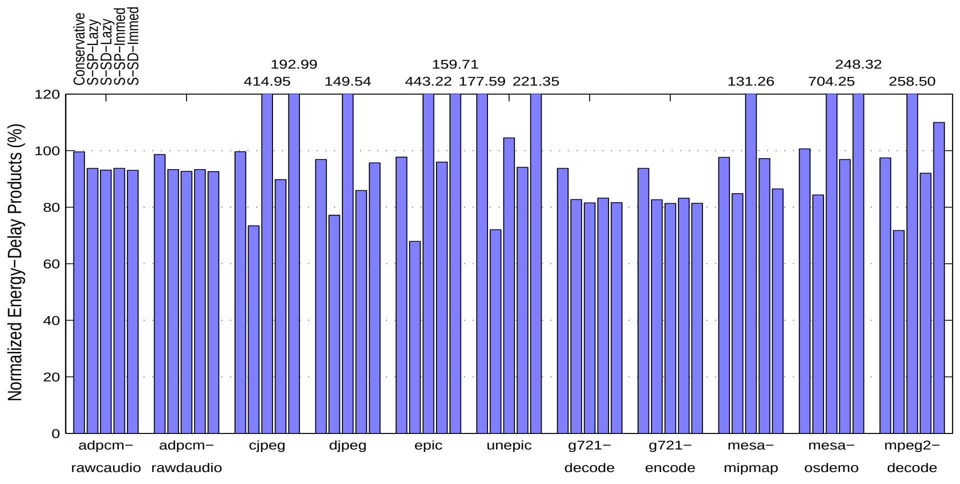

Figure 7: **Normalized energy-delay products of our optimization strategies. (MediaBench)**

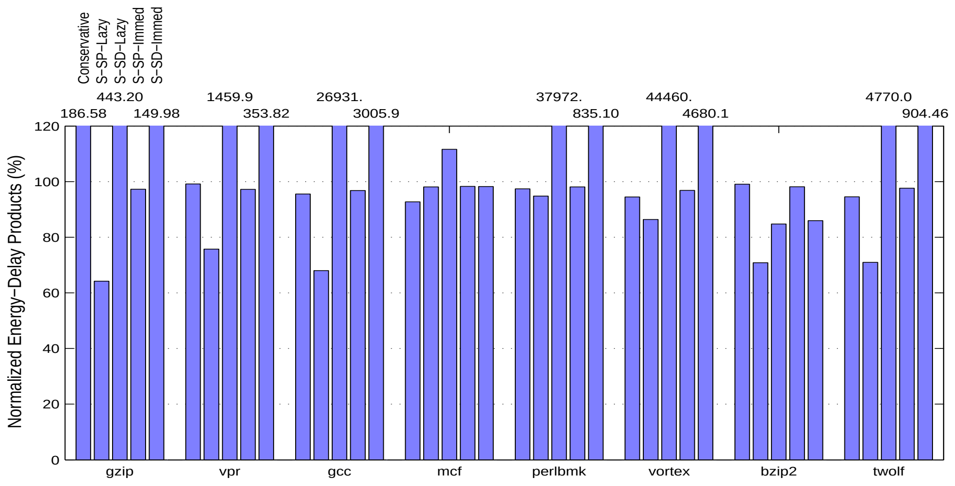

Figure 8: **Normalized energy-delay products of our optimization strategies. (SPEC CINT2000)**

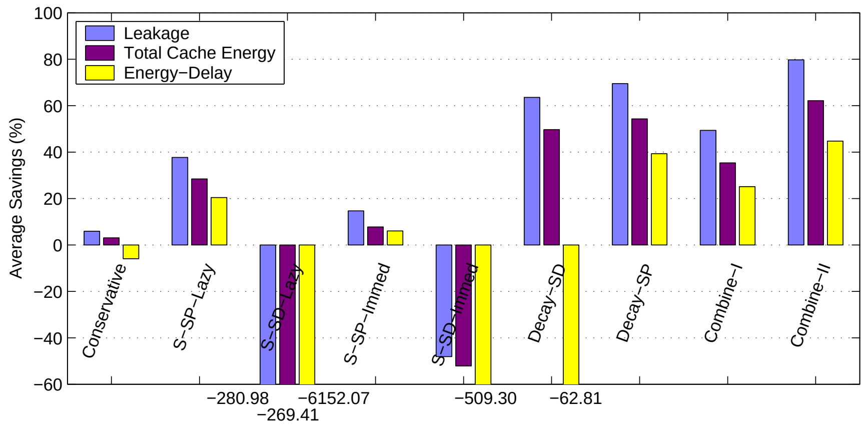

Figure 9: **Average savings (over all benchmarks) for different optimization strategies.**

execution cycles and the additional leakage expended in the other parts of the processor during these additional cycles. We assume conservatively that the contribution of the rest of the processor (other than the cache subsystem) to the leakage energy is 30% in our calculations. The energy-delay product is a suitable metric that allows to evaluate the impact of an optimization on both the performance and energy. The results given in Figure 7 and Figure 8 are the normalized energy-delay products (with respect to the original cache management without any leakage energy control). It is easy to see that S-SD-Lazy and S-SD-Immed do not perform well due to frequent main memory visits resulting from L2 misses. We observe, however, that the S-SP-Lazy strategy reduces the energy-delay product by 20.4%, on the average. Apart from S-SP-Lazy, only S-SP-Immed improves the energy delay product (6.0% on the average). State-destroying optimization strategies, on the other hand, increase energy-delay product by 5.3% (Conservative), 6152.1% (S-SD-Lazy), and 509.3% (S-SD-Immed). Based on these results, we can conclude that working with a state-preserving mode is extremely important to improve both energy and the energy-delay product. The first five groups of bars in Figure 9 show the average values (percentage improvements) for our five optimizations across all benchmark programs for leakage energy, overall cache energy, and energy-delay product. The last four groups of bars are discussed in the next section.

4. Comparison and Integration with Other Strategies

In [10], Kaxiras et al. present a leakage energy reduction technique for cache memories. This technique, called *cache decay*, is based on the idea that a cache block that is not used for a sufficiently long period of time can be considered *dead*. More specifically, with each cache block, they associate a small 4-state FSM (finite state machine). The FSM steps through these states as long as the cache block is not accessed. When the last state is reached, the cache block is turned off.

To compare this technique with our approach, we implemented cache decay in SimpleScalar [25] and performed experiments. The first implementation (called `Decay-SD`) is a straightforward extension of their approach to a cache hierarchy (instead of just the L1 cache). Specifically, we applied the cache decay method to both L1 and L2 using the state-destroying leakage saving technology. Then, we further enhanced this scheme by employing the state-preserving strategy in both L1 and L2. In this second implementation (called `Decay-SP`), the L2 cache is energy-managed

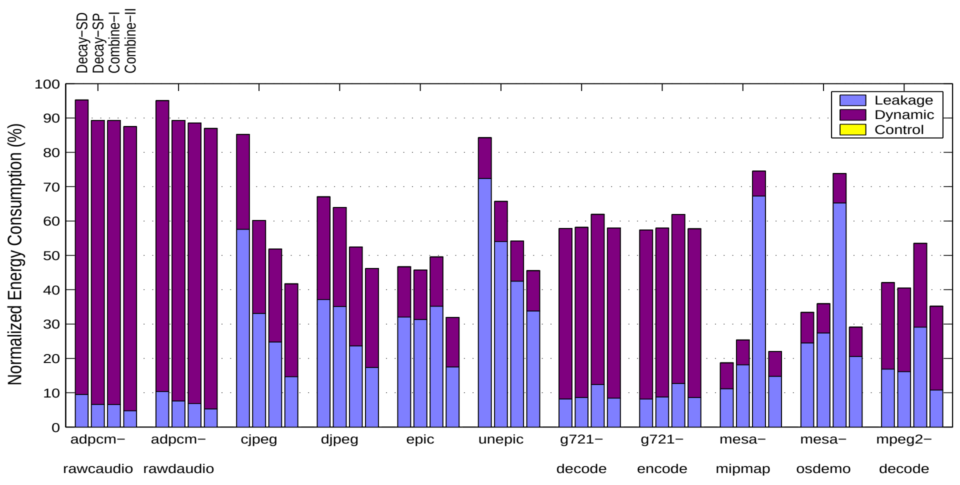

Figure 10: **Normalized energy consumption of cache decay and combined strategies. (MediaBench)**

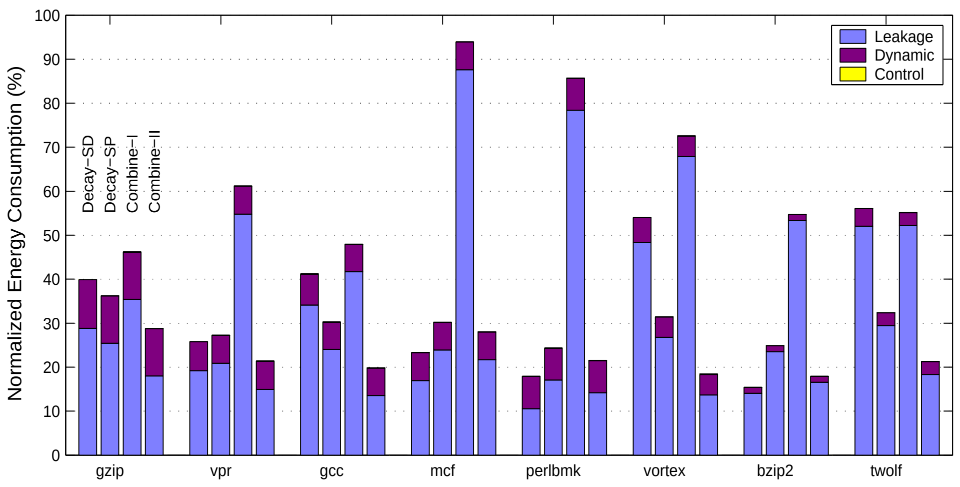

Figure 11: **Normalized energy consumption of cache decay and combined strategies. (SPEC CINT2000)**

at the subblock granularity and the FSM is used to transition L2 subblocks into a state-preserving mode (as opposed to the state-destroying mode in Decay-SD). In both of these implementations, we used the threshold values used in [10] (i.e., 10K cycles for L1 and 1M cycles for L2).

In addition to these two strategies, we implemented two strategies that combine the cache decay scheme with our optimization strategy. `Combine-I` corresponds to a strategy where L1 leakage energy is optimized using cache decay method, whereas the L2 cache energy is optimized using our S-SP-Lazy strategy. Finally, `Combine-II` employs cache decay for L1, but uses *both* cache decay (as in Decay-SP) and our S-SP-Lazy strategy for L2. In both `Combine-I` and `Combine-II`,

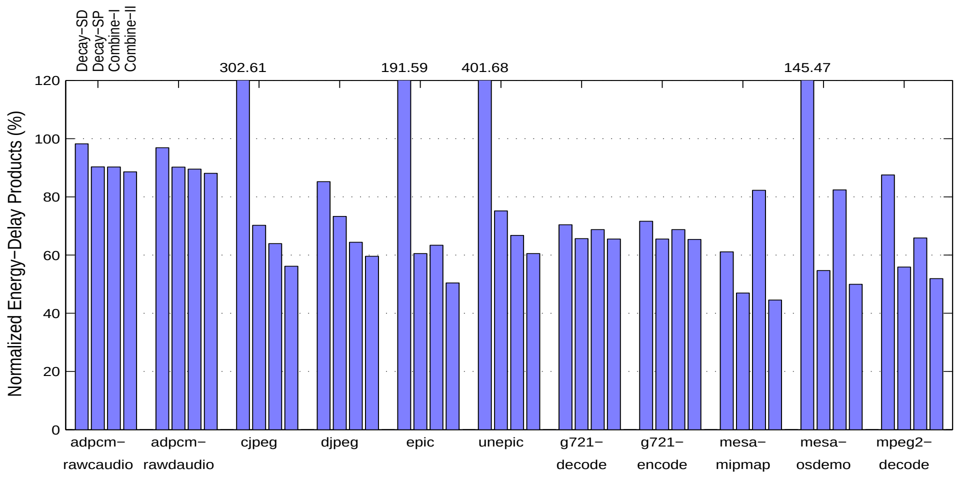

Figure 12: **Normalized energy-delay products of cache decay and combined strategies. (MediaBench)**

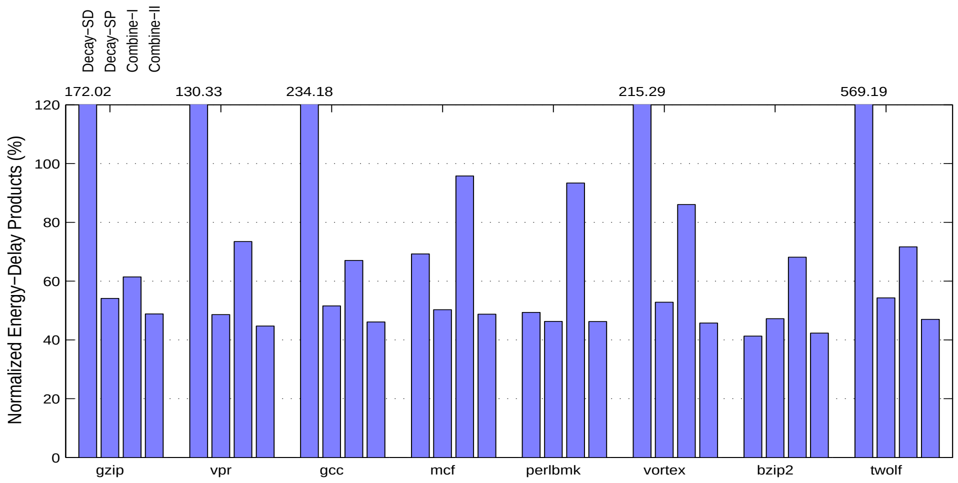

Figure 13: **Normalized energy-delay products of cache decay and combined strategies. (SPEC CINT2000)**

state-preserving mechanism is employed in L1 and L2. The reason that we used S-SP-Lazy in these last two versions (instead of other speculative strategies) is that it performs better than others as shown through our experimental results discussed earlier.

Figures 10 and Figures 11 show the normalized energy consumptions. Figures 12 and Figures 13 show the normalized energy-delay products, respectively, for these last four strategies mentioned above. It can be observed that Decay-SD performs quite well and improves leakage energy consumption and overall cache energy by 63.6% and 49.7%, on the average. However, it increases execution cycles as, under this optimization scheme, it is possible that a cache block can be destroyed in L1 as well as in L2. Consequently, it incurs frequent main memory accesses, thus degrading the energy-delay product by 62.8%. In most cases, Decay-SP improves over Decay-SD in both energy and energy-delay product. It improves leakage energy, total cache energy, and energy-delay product by 69.5%, 54.3%, and 39.3%, respectively. This is because in Decay-SP, both L1 and L2 cache management do not destroy data, thereby preventing frequent main memory accesses. Also, as compared to Decay-SD, it manages L2 leakage energy in subblock granularity. Recall that S-SP-Lazy's leakage energy, total cache energy, and energy-delay product improvements were 37.6%, 28.5%, and 20.4%, respectively. While Decay-SD provide a better energy reduction over S-SP-Lazy, it suffers from long execution times. Decay-SP, however, performs better than S-SP-Lazy in all aspects. It should also be stressed that while S-SP-Lazy targets only the lines in L2 cache that are moved into L1 cache, Decay-SD and Decay-SP target both caches and hence they have potentially larger optimization scope. In fact, comparing the savings *only* in the L2 cache shows that S-SP-Lazy, Decay-SD, and Decay-SP reduce leakage energy consumption by 48.2%, 59.7%, and 69.2%.

Combine-I improves the unoptimized leakage energy by 49.3%, overall cache energy by 35.3%, and energy-delay product by 25.1%. But the savings are less than that of Decay-SP. This is because S-SP-Lazy targets only the lines in L2 cache that are moved into L1 cache and does not target the cache blocks which just stay in L2 cache for a long period of time (Decay-SP does). Therefore, we implement Combine-II, which employs both S-SP-Lazy and cache decay in L2 cache. Combine-II generates the best energy results among these optimization strategies. As compared to the unoptimized case, it improves leakage and overall cache energy by 79.7% and 62.1%, respectively. These energy benefits are due to its aggressive optimization strategy in L2. More specifically, the two different methods (S-SP-Lazy and cache decay) complement with each other to optimize L2 energy. The last four groups of bars in Figure 9 summarize the average improvements for the four optimization strategies discussed in this section from the leakage energy, overall cache energy, and energy-delay product perspectives.

5. Sensitivity Analysis

To measure the robustness of the energy optimization strategies studied in this paper, we also measured their sensitivity to different parameters. In particular, we measured the energy savings when cache configuration parameters (cache capacity, associativity, and block size), relative magnitudes of leakage and dynamic energies, leakage saving factor in the state-preserving mode, the ratio of L2 subblock and L1 block leakage, and the effect of different reactivation time.

We first focus on one benchmark (`epic`) and vary the L1 and L2 cache parameters. The trends observed in other benchmarks are similar to that of `epic`, so they are not included. Also, we focus only on S-SP-Lazy and Combine-II strategies. Figure 14 gives the cache energy consumptions. Each cache configuration is denoted as

(L1-Size,L1-Associativity,L1-Block-Size;L2-Size,L2-Associativity,L2-Block-Size),

with (32KB,2,32;1MB,2,128) being our default configuration. Note that in each experiment L1 instruction and L1 data caches have the same configuration. Each bar in this figure represents energy consumption normalized with respect to the energy consumption of the simulation without any leakage control with the default cache configuration. First we observe that energy savings are obtained across different configuration. Further, when the L2 cache size increases, we witness a

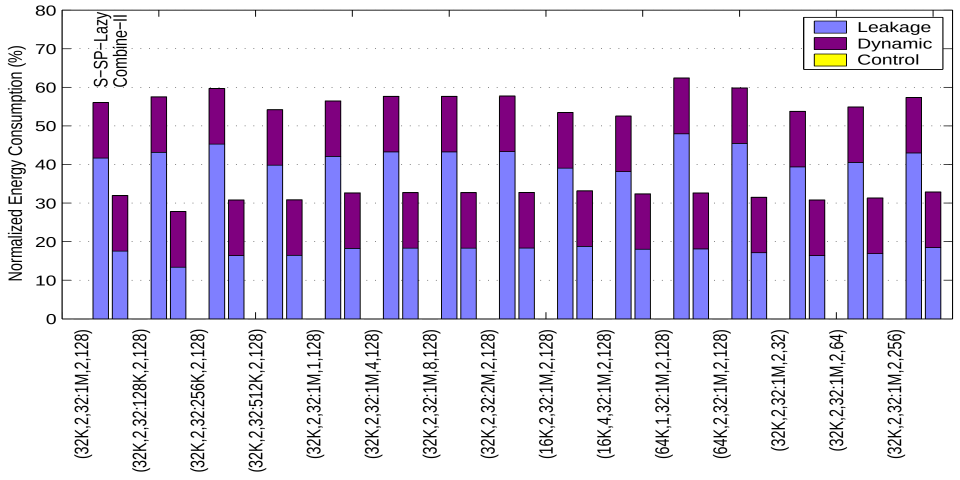

Figure 14: **Normalized energy consumptions for different cache hierarchy configurations (`epic`).**

decrease in energy savings. The reason for this is that when the L2 capacity is higher, the number of L2 subblocks that are in active or state-preserving mode increases.

We next modify the relative magnitude of leakage energy per cycle with respect to dynamic energy per access. Specifically, we assume that the leakage energy per cycle of the entire L1 cache is equal to *half* of the dynamic energy consumed per access. So, this gives more weight to dynamic energy. Figure 15 gives the average improvements in this case for leakage energy, overall cache energy, and energy-delay product. As in the previous case (Figure 9), Combine-II results in the best energy behavior (79.73% improvement in leakage and 55.0% improvement in total cache energy) and energy-delay product (40.3%). We see that, as expected, the leakage energy improvements do not change significantly. But, since the impact of dynamic energy is increased, we witness a decrease in overall cache energy savings compared to Figure 9. In addition, in this case, the overhead cost (from the dynamic energy viewpoint) (due to our leakage optimization strategies) is higher. We observe a similar reduction in energy-delay product savings. This is a direct result of the decrease in overall energy savings.

We now modify the leakage saving factor when state-preserving leakage control mode is employed. Recall that we earlier assumed that, in each cycle, the state-preserving strategy consumes 10% of the original (active) per cycle leakage energy. We call this figure *leakage saving factor.* In this set of experiments, we changed this figure to 30%. All other simulation parameters are the same as in our base configuration. The results given in Figure 16 (average values over all benchmark codes) reveal that, even in this case, the state-preserving strategy is superior to the state-destroying one. In particular, both S-SP-Lazy and S-SP-Immed improves leakage energy, overall energy, and energy-delay product. However, the energy savings due to preserving state are not as good. For instance, with this new setup, S-SP-Lazy and S-SP-Immed improve overall cache energy by 29.2% and 11.4%, respectively. In contrast, the corresponding numbers when we employed a leakage saving factor of 10% were 37.7% and 14.6%.

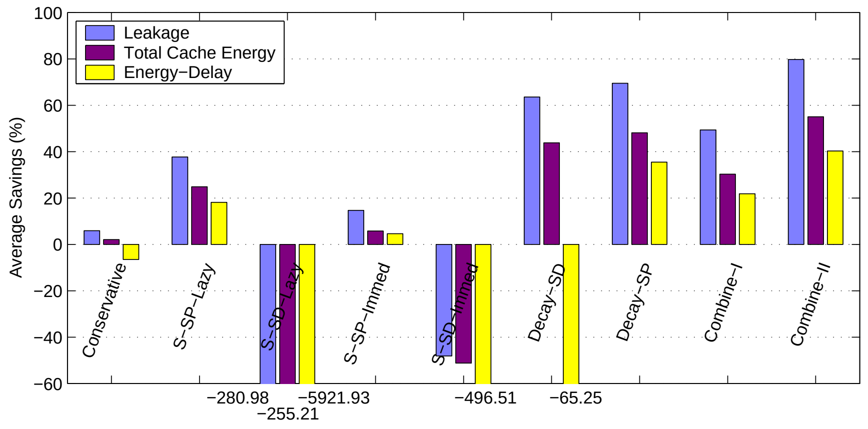

Figure 15: **Sensitivity to the relative magnitude of dynamic versus leakage: average leakage energy, total cache energy, and energy-delay product improvements for different optimization strategies.(over all benchmarks)**

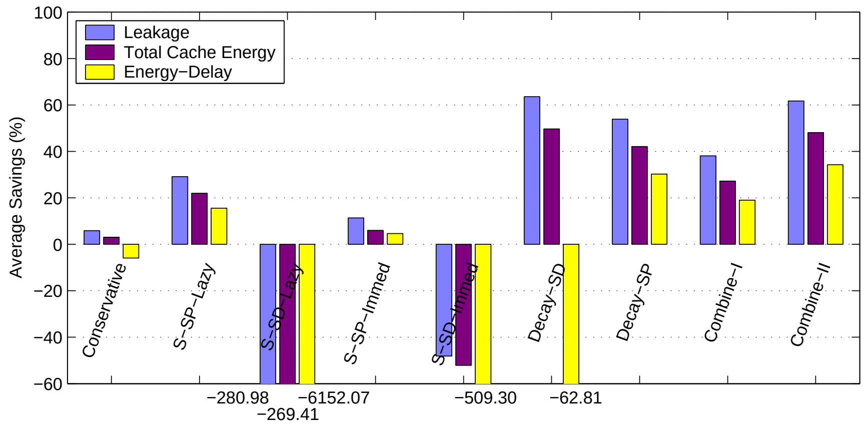

Figure 16: **Sensitivity to the leakage saving factor: % improvements in average leakage energy, total cache energy, and energy-delay product for different optimization strategies. (over all benchmarks)**

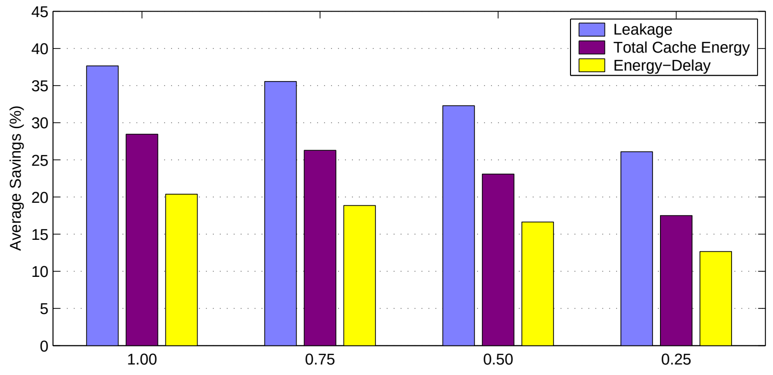

Figure 17: **Sensitivity to the ratio of L2 subblock and L1 block leakage in S-SP-Lazy method.(over all benchmarks)**

We then modify the ratio between leakage of L2 subblock and leakage of L1 block. Recall that the default value was 1.00. But, using higher threshold voltage can let L2 cache consume less leakage energy at the expense of extra L2 latency. Figure 17 shows the improvements of S-SP-Lazy method in leakage and total cache energy for different ratios. We can conclude that our strategy still brings benefits when the ratio of L2 vs L1 is reduced.

The time to reactivate a cache line in state-preserving or state-destroying mode to its normal state depends on the actual circuit implementation. So far in this paper, we have used the voltage scaling approach that incurs only a single cycle penalty for this reactivation. We consider an alternate implementation using a modified gated-Vdd technique to study the influence of this parameter.

We custom-designed a 16-bit array of memory cells in 0.07 micron technology and performed circuit simulation using HSPICE. We observed that the state of these cells were maintained from a supply voltage of 1.0V down to 120mV. In order to achieve a 120mV voltage, a sized NMOS switch was introduced between the ground rail and the memory cell. Using the NMOS switch can reduce both bitline and cell leakage. When the power-switch is turned-on, a normal supply voltage is provided to the circuit. However, when the power-switch is turned-off, the ground level rises to 0.88V from 0V. Note that this is achieved by having an appropriately sized power-switch that has a controlled leakage to provide the required minimum supply voltage. A 0.68μm/0.07μm (width/length) NMOS device (with a threshold voltage of 200mV) was used as a power-switch along with each memory array to achieve the required supply voltage of 120mV (See Figure 18). We observe from our simulations that the overall leakage of the memory array can be reduced to 4% of original leakage using the state-preserving mechanism. However, in order to activate a cache line in state-preserving mode to a normal mode, our simulations indicate that a 19ns latency is required for the ground level to settle back to 0V. This would incur a 38 cycle penalty as compared to the 1 cycle latency required for voltage settling in the circuit described in Section 2.

Thus, it is clear that the circuit choices can influence the latency for reactiving a cache block in standby mode. To study this impact, in Figure 19 we simulate the `epic` with different reactivation latency 50 cycles, 20 cycles, and 1 cycle, and compare the effect on energy consumption and energy-delay product of four strategies. Comparing with long reactivation latency, small reactivation latency performs very small increase in leakage and total cache energy savings, but it

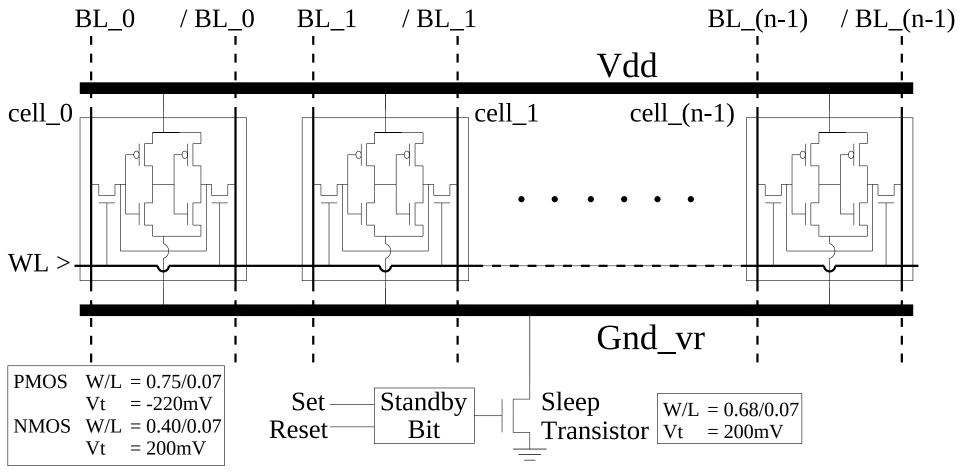

Figure 18: **An L2 subblock augmented with leakage control mechanism.**

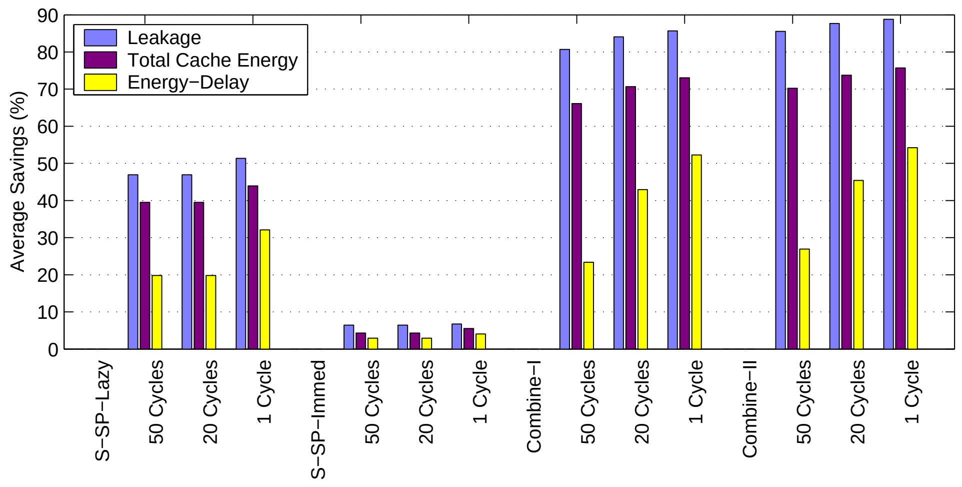

Figure 19: **Sensitivity to the reactivation time: % improvements in average leakage energy, total cache energy, and energy-delay product for `epic`.**

has significant improvements on energy-delay product for S-SP-Lazy, Combine-I, and Combine-II strategies.

6. Conclusions and Future Work

Duplication of data and instructions at different levels of memory hierarchy is costly from the leakage energy perspective. This paper first examined a leakage control mechanism that can preserve the state of the memory cell. Using this state-preserving leakage control mechanism and a state-destroying leakage control mechanism, we investigated five different strategies to put L2 subblocks that hold duplicate copies of L1 blocks in energy saving states (leakage control modes). These strategies differed from each other with respect to the circuit type that they employ (state-destroying versus state-preserving), whether they conservatively or speculatively turn off L2 subblocks, and the time that the L2 subblocks are reactivated. Our experimental results indicated that the best strategy (S-SP-Lazy) in terms of energy and energy-delay product is to place the L2 subblock into a state-preserving leakage control mode as soon as its contents are moved to L1 and to reactivate it only when it is accessed. We then integrated this strategy with a previously proposed optimization scheme, called cache decay, and showed that the integrated strategy generates better energy results.

In addition, we conducted a sensitivity analysis by changing several simulation parameters such as cache configuration, leakage saving factor, the relative magnitude of per cycle leakage consumption with respect to per access dynamic energy consumption, and the reactivation time. We believe that this work is a step towards achieving our eventual goal of optimizing energy consumption at different levels of the architecture without sacrificing too much performance.

This work can be extended in multiple ways:

- *More powerful combined optimization strategies:* In our current implementations of combined strategies, a given data item can be placed into energy-saving mode in both the caches at the same time. This incurs a performance penalty when such data is subsequently accessed. We plan to investigate mechanisms that would maintain at least one active copy of data in either L1 or L2 at any given time. We also want to perform experiments with strategies that combine adaptive cache decay [10] with our speculative schemes.

- *Combining state-preserving and state-destroying strategies:* In some cases, it might be beneficial to use state-preserving and state-destroying strategies together. For example, in bringing a data from L2 to L1, we can first place the corresponding L2 subblock into the state-preserving state. After some time, if the subblock is not activated, we can switch the subblock's state to the state-destroying one. Note that this would require changes to our current circuit implementation to enable dynamic switching between state-preserving and state-destroying modes.

- *Software-based leakage optimization:* We are in the process of building a compiler-based strategy that tries to locate the last use of program variables. After determining the last use of a variable, it modifies the corresponding load instruction such that, when executed at runtime, the modified load turns off the cache block that holds the variable. Obviously, the success of such a strategy depends strongly on compiler's ability of identifying last use of variables. We have recently implemented a compiler based leakage control for the instruction cache [29].

- *Integrating hardware-based and software-based strategies:* Our long term plan is to exploit both hardware-based and compiler-based strategies in a unified optimization framework. This framework will analyze the last use of variables and, depending on the result of this analysis, will select a suitable combination of optimizations.

- *Sensitivity to soft errors:* A detailed analysis of the impact of soft errors when using low voltages to maintain memory state is beyond the scope of this paper and is part of our ongoing research. Initial research of this analysis is available in [24].

References

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

[2] K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in low-power RAM circuit technologies," *Proceedings of IEEE*, vol. 83, pp. 524–543, Apr. 1995.

[3] S. Kim, N. Vijaykrishnan, M. Kandemir, A. Sivasubramaniam, M. J. Irwin, and E. Geethanjali, "Power-aware partitioned cache architectures," in *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED'01)*, pp. 64–67, 2001.

[4] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proceedings 1999 international symposium on Low power electronics and design (ISLPED'99)*, pp. 273–275, 1999.

[5] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pp. 54–65, 2001.

[6] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture (MICRO-32)*, pp. 248–259, 1999.

[7] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO-30)*, pp. 184–193, 1997.

[8] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture (MICRO-33)*, pp. 214–220, 2000.

[9] S. 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 Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, pp. 147–157, 2001.

[10] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings of the 28th annual international symposium on on Computer architecture (ISCA-28)*, pp. 240–251, 2001.

[11] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko, "Tuning garbage collection in an embedded java environment," in *Proceedings of Eighth International Symposium on High-Performance Computer Architecture (HPCA-8)*, pp. 80–91, 2002.

[12] P. R. V. d. Meer and A. V. Staveren, "Standby-current reduction for deep sub-micron VLSI CMOS circuits: smart series switch," in *the ProRISC/IEEE Workshop*, pp. 401–404, Dec. 2000.

[13] B. Nikolic, “State-preserving leakage control mechanisms,” *Gigascale Silicon Research Center Annual Report*, Sept. 2001.

[14] A. Agarwal, H. Li, and K. Roy, “Drg-cache: a data retention gated-ground cache for low power,” in *Proceedings of the 39th conference on Design automation (DAC-39)*, pp. 473–478, 2002.

[15] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, “Adaptive mode control: A static-power-efficient cache design,” in *the 10th International Conference on Parallel Architectures and Compilation Techniques (PACT’01)*, pp. 61–70, Sept. 2001.

[16] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy caches: simple techniques for reducing leakage power,” in *Proceedings of the 29th annual international symposium on Computer architecture (ISCA-29)*, pp. 148–157, 2002.

[17] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, “Drowsy instruction caches - leakage power reduction using dynamic voltage scaling,” in *Proceedings of the 33th annual ACM/IEEE international symposium on Microarchitecture (MICRO-35)*, Nov. 2002.

[18] H. Qin and J. Rabaey, “Leakage suppression of embedded memories,” *Gigascale Silicon Research Center Annual Review*, 2002.

[19] 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 (ISCA-29)*, pp. 137–147, 2002.

[20] N. P. Jouppi and S. J. E. Wilton, “Tradeoffs in two-level on-chip caching,” in *Proceedings of the 21ST annual international symposium on Computer architecture (ISCA-21)*, pp. 34–45, 1994.

[21] T. May and M. Woods, “Alpha-particled-induced soft errors in dynamic memories,” *IEEE Trans. on Electron Devices*, vol. ED-26, Jan. 1979.

[22] “Berkeley predictive model. http://www-device.eecs.berkeley.edu.”

[23] L. Li, I. Kadayif, Y.-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam, “Leakage energy management in cache hierarchies,” in *the 11th International Conference on Parallel Architectures and Compilation Techniques (PACT’02)*, pp. 131–140, Sept. 2002.

[24] V. Degalahal, N. Vijaykrishnan, and M. J. Irwin, “Analyzing soft errors in leakage optimized sram designs,” in *Sixteenth International Conference on VLSI Design*, Jan. 2003.

[25] D. C. Burger and T. M. Austin, “The SimpleScalar tool-set, Version 2.0,” Tech. Rep. 1342, Dept. of Computer Science, UW, June 1997.

[26] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems,” in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO-30)*, pp. 330–335, 1997.

[27] “Spec cpu2000 benchmark. http://www.spec.org/.”

[28] R. Cooksey and D. Grunwald, “Characterization of the spec2000 benchmark suite. http://www.cs.colorado.edu/ rcooksey/pubs.html,” 2001.

[29] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, “Compiler-directed instruction cache leakage optimization,” in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture (MICRO-35)*, Nove. 2002.

[30] L. Benini and G. de Micheli, “System-level power optimization: techniques and tools,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 2, pp. 115–192, 2000.

[31] S. Borkar, “Design challenges of technology scaling,” *IEEE Micro*, vol. 19, pp. 23–29, July 1999.

[32] J. A. Butts and G. S. Sohi, “A static power model for architects,” in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture (MICRO-33)*, pp. 191–201, 2000.

[33] 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 (ISCA-27)*, pp. 83–94, 2000.

[34] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[35] R. Gonzalez and M. Horowitz, “Energy dissipation in general purpose microprocessors,” *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1277–1284, Sept. 1996.

[36] H. Kawaguchi, K. Nose, and T. Sakurai, “A super cut-off CMOS scheme for 0.5V supply voltage with pico-ampere standby current,” *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 1498–1501, Oct. 2000.

[37] T. Kuroda and T. Sakurai, “Threshold-voltage control schemes through substrate-bias for low-power high-speed CMOS LSI design,” *Journal of VLSI Signal Processing Systems*, vol. 13, pp. 191–201, Aug. 1996.

[38] S. Mutoh and et al, “1-V power supply high-speed digital circuit technology with multi-threshold-voltage CMOS,” *IEEE Journal of Solid State Circuits*, vol. 30, pp. 847–854, Aug. 1995.

[39] J. Rabaey, *Digital integrated circuits: a design perspective*. Prentice Hall Inc. online revisions of new chapters.
http://bwrc.eecs.berkeley.edu/Classes/ICDesign/EE141f00/notes.html.

[40] Y. Ye, S. Borkar, and V. De, “A new technique for standby leakage reduction in high-performance circuits,” in *the Symposium on VLSI Circuits*, pp. 40–41, 1998.