Leakage Energy Management in Cache Hierarchies *

L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin and A. Sivasubramaniam
Microsystems Design Lab, Pennsylvania State University
{*lili, kadayif, ytsai, vijay, kandemir, mji, anand*}*@cse.psu.edu*

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. 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 media and array-dominated applications, 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.

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 [5]. 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 two reasons: 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 [18, 2]. Among these are partitioning large caches into smaller structures to reduce the dynamic energy [9] and the use of a memory hierarchy that attempts to capture most accesses in the smallest size memory. 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 [19] to dynamically shutdown portions of the cache. This technique was applied at a cache block granularity in [12] and used in conjunction with software to remove dead objects in [6]. 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 [15, 16]. 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 [19], the cache size is reduced (or increased) dynamically to optimize the utility of the cache. In [12], the cache block is supply-gated if it has not been accessed for a period of time. In [20], hardware tracks the hypothetical miss rate and the real miss rate by keeping tag line active when deactivating a cache line. And then the turn-off interval can be dynamically adjusted based on such information. In [7], dynamic supply voltage scaling is used to reduce the leakage in the unused portions

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

of the memory. In contrast to the other schemes, it also preserves data when in low leakage mode. Our technique also supports a data-preserving leakage control and is based on a different circuit implementation that has been verified using circuit simulation. The focus in [8] 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. 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 [10]. Our technique mimics exclusion by putting a duplicated copy to sleep mode.

In this paper, we make the following major contributions:

- We present a circuit-level mechanism to implement state-preserving (data-retaining) leakage control at an L2 subblock granularity and compare its effectiveness to state-destroying leakage control.
- Based on these state-preserving and state-destroying mechanisms, we describe five leakage reduction strategies that exploit data duplication in the cache hierarchy. Using a set of media and array-dominated applications, 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. Finally, in Section 5, we summarize our major contributions .

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 [5]. 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. However, the addition of the series switch in the circuit between the supply and the output affects performance by bringing an additional delay. The magnitude of this delay depends on the sizing of series switch and, with a larger switch, it can be reduced. Additional dynamic energy is required to turn this power-switch on and off. There are several variants of this underlying scheme that have been implemented including the super cut-off CMOS [11] and MTCMOS [5] techniques. Instead of adding this switch between the supply and the circuit, a switch could also be added between the ground and the circuit to reduce leakage as in the gated-Vdd scheme [19]. In all these leakage control schemes, no particular attention has been paid to preserving the state of the circuit when it is put into the sleep state.

The key to maintaining the state of a static memory cell is the operation of the two-chain inverter loop that retains the state. This inverter chain can maintain the state even when the supply voltage of the memory cell is dropped to a very small value, even lower than the threshold voltage of the device. This state-preserving action happens as the voltage transfer characteristics of the inverter chain are valid until the supply voltage drops to about 100 mV [17]. Until this point, the sub-threshold currents are sufficient to switch

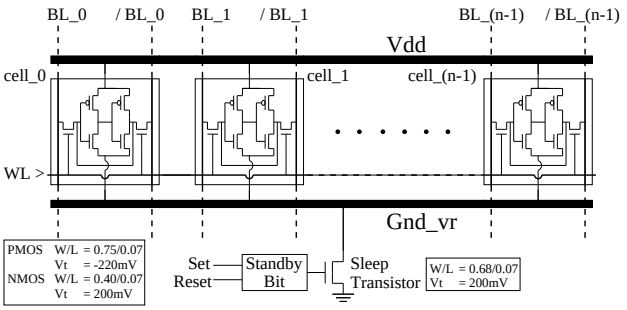

Figure 1. An L2 subblock augmented with leakage control mechanism.

the gate between low and high levels and provide sufficient gain to produce acceptable voltage transfer characteristics for preserving the state. This observation coupled with the fact that the leakage energy reduces with decrease in supply voltage can be used to realize a circuit that achieves state-preserving leakage control [16].

In order to experimentally verify this behavior, we custom-designed a 16-bit array of memory cells in 0.07 micron technology (Berkeley predictive model [1]) 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. Note that gate leakage is not supported by our leakage models. 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 1). However, the leakage reduction provided by this state-preserving technique is observed to be less than that when completely dropping the virtual supply voltage to zero. This is because leakage is required to maintain the state. We observe from our simulations that the overall leakage of the memory array can be reduced to 4% of original leakage using our state-preserving mechanism. Thus, in this work, we conservatively 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. We also observe that the performance penalty is negligible as the resistance of the sized series NMOS switch is very small.

In our implementations, for both state-preserving and state-destroying, we turn off the power-switch of the L2 cache subblock during standby and turn it on when the cache subblock needs to be accessed. We activate and deactivate the power-switch as shown in Figure 1 using the standby bit. Whenever the standby bit is set to zero, the supply voltage drops to 120 mV (resp. $\sim$0 V) to move the circuit to a standby state-preserving (resp. state-destroying) leakage control mode. This bit is set to one for the circuit to 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.

There is additional dynamic energy consumed in switching these transistors that is reflected as *control energy* (also called *control overhead*) in our experiments. This control energy is determined by the gate capacitance of the power-switch and the associated wiring used to control the power-switch. Further, it takes a finite time for the virtual ground line to settle after the power-switch is transitioned. In our design, a 0.68μm/0.07μm (width/length) NMOS switch is connected to every 16 cells to reduce area overhead to 2.3%. A 19ns (38 cycles) settling time is observed for our design. In our experiments, we conservatively assume a 50 cycle virtual supply settling time based on this.

There are additional constraints imposed by the state-preserving leakage control. Since, the voltage used to maintain the state is very small (120 mV), just the effect of turning on the power-switch rapidly can cause the memory cell to flip states due to coupling between the input and the output. This requirement imposes a minimum rise time for the control input at the power-switch and is of the order of 1V/10ns [16]. This requirement is less stringent than the settling time for the virtual supply voltage. 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 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 [14]. 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.

If an L2 cache subblock 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. However, if, after being put into the low-power mode, the cache subblock 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 subblock 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 subblock in question. This is, obviously, a characteristic of application access pattern and cache hierarchy configuration.

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.
- `Speculative-I:` In this strategy, when data is brought from L2 to L1, the corresponding L2 subblock is put in a state-preserving leakage control mode.

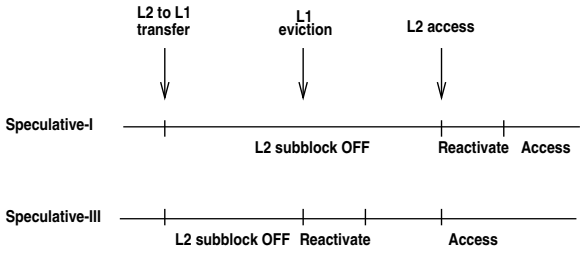

Figure 2. Comparison of Speculative-I and Speculative-III.

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

- `Speculative-II:` This strategy is similar to Speculative I, 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 Speculative-I (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 Speculative-I as it needs to access the off-chip memory (as opposed to Speculative-I which simply reactivates the L2 subblock).
- `Speculative-III:` This is also similar to Speculative I except that the L2 subblock is reactivated whenever the corresponding L1 cache block needs to be replaced. This early reactivation (as compared to Speculative I where reactivation occurs only when the L2 block is accessed) can reduce energy savings compared to Speculative-I. 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 Speculative-I 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 Speculative-III, it is reactivated when the L1 eviction occurs. Consequently, in Speculative-III, the L2 sub-

Table 1. Proposed leakage energy saving strategies.

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
`Speculative-I`	when L2 subblock is moved to L1	state-preserving	when accessed
`Speculative-II`	when L2 subblock is moved to L1	state-destroying	when accessed
`Speculative-III`	when L2 subblock is moved to L1	state-preserving	when L1 block is evicted
`Speculative-IV`	when L2 subblock is moved to L1	state-destroying	when L1 block is evicted

block reactivation time can be hidden.

- `Speculative-IV:` This strategy is similar to Speculative II 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 Speculative II are similar to those of Speculative III with respect to Speculative I. Similarly, its advantages/disadvantages compared to Speculative III are similar to those of Speculative II compared to Speculative I.

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.

3.2. Simulation Parameters and Benchmarks

We used Simplescalar 3.0 [4] 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 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 [13] and FP-intensive routines from Spec and Perfect Club benchmarks. We selected these two groups of codes as they represent different access patterns. For each code in our experimental suite, the simulations are run to completion. The important characteristics of these benchmarks are

Table 2. Our base configuration.

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 unified, 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
Virtual Supply Settling Time	50 cycles
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

listed in Table 3. The fifth and sixth 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 (65% of the cache hierarchy energy on the average) and expend a large fraction of this leakage energy (74.2% on the average [1]) in L2 due to its much larger capacity. Conse-

[1] This is when no leakage optimization is applied

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

Benchmark	Source	Input	Execution Cycles (in millions)	Cache Energy		L2 Leakage Contribution
				Leakage (mJ)	Dynamic (mJ)	
adpcm-rawcaudio	MediaBench	clinton.pcm	4.74	1.88 (17.46%)	8.90 (82.54%)	57.28%
adpcm-rawdaudio	MediaBench	clinton.adpcm	4.00	1.54 (18.37%)	6.85 (81.63%)	56.41%
cjpeg	MediaBench	testimg.ppm	7.69	54.25 (72.97%)	20.09 (27.03%)	77.76%
djpeg	MediaBench	testimg.jpg	2.93	13.61 (71.22%)	5.49 (28.78%)	68.77%
epic	MediaBench	test_image.pgm	21.33	352.65 (85.62%)	59.20 (14.38%)	90.53%
unepic	MediaBench	test.image.pgm.E	5.53	75.31 (88.29%)	9.98 (11.71%)	88.88%
g721-decode	MediaBench	clinton.g721	119.09	338.40 (50.47%)	332.10 (49.53%)	55.14%
g721-encode	MediaBench	clinton.pcm	123.98	353.54 (50.85%)	341.75 (49.15%)	55.30%
mesa-mipmap	MediaBench	-	37.09	1032.67 (92.75%)	80.75 (7.25%)	94.02%
mesa-osdemo	MediaBench	-	11.97	315.95 (91.48%)	29.40 (8.52%)	93.77%
mpeg2-decode	MediaBench	mei16v2.m2v	65.36	638.12 (75.62%)	205.71 (24.38%)	84.30%
tomcatv	Spec	273.68KB	11.27	8.81 (71.57%)	3.50 (28.43%)	86.60%
tsf	Perfect Club	50.86KB	10.56	1.00 (63.92%)	0.56 (36.08%)	61.65%
vpenta	Spec	609.64KB	12.29	22.22 (88.00%)	3.02 (12.00%)	90.16%
wss	Perfect Club	31.89KB	11.03	1.63 (41.52%)	2.29 (58.48%)	52.16%

quently, we can expect large leakage energy savings using our strategies. Most of the energy results given in following subsections are results *normalized* with respect to the values in the fifth and sixth columns of Table 3.

3.3. Limits in Energy Savings

We first present in Figure 3 the results of an optimal (perfect) energy optimization strategy (from the leakage perspective) in which the L1 and L2 cache blocks are activated only when they are accessed; otherwise, they are turned off (in state-destroying sense) to save leakage energy. We assume that the reactivation cost in this optimal scheme (from both performance and energy perspectives) is zero. We observe from these results that an optimal leakage optimization strategy can reduce the original leakage consumption to 6.6% on the average. Also, the optimal leakage consumption is always below 10% of the original (9.3% being the largest value). These results serve as a bar against which our five leakage optimization strategies can be compared. It should be noted that in practice it is not possible to realize these optimal leakage savings as reactivation costs are not zero. It should also be mentioned in obtaining these results it is assumed that no additional dynamic energy optimization strategy is employed.

3.4. Impact of Our Strategies

Figure 4 shows 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). 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

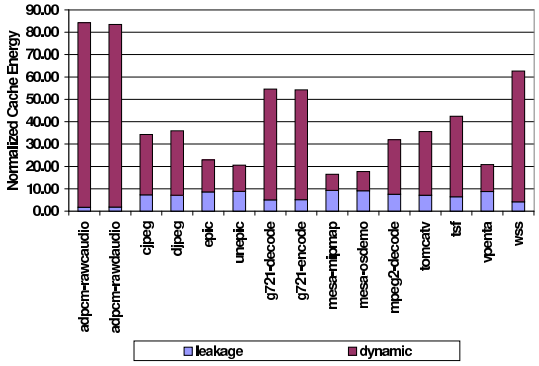

Figure 3. Optimal energy consumption (from the leakage energy perspective).

in the Conservative strategy. This makes sense as in this strategy the control overhead is directly proportional to the number of L1 writes.[2] Second, among our five strategies, Speculative-I generates the best energy results. It reduces leakage energy consumption by 37.1% on the average and overall cache energy (including dynamic energy and control overhead as well) by 23.3% on the average. On the other hand, the average leakage (resp. the average overall energy) improvements due to Conservative, Speculative-II, Speculative-III, and Speculative-IV are 8.0% (resp. 1.8%), -68.4% (resp. -63.9%), 22.8% (resp. 11.7%), and 15.2% (resp. 4.9%), respectively. (A negative value indicates an increase in energy consumption).

We now explain why these different strategies performed the way they did. Comparing Speculative-I and Speculative-II, 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 Speculative-I but is

[2] In our codes, the ratio of *L1 Writes/Total L1 Accesses* varies between 10.1% (epic) and 59.2% (wss).

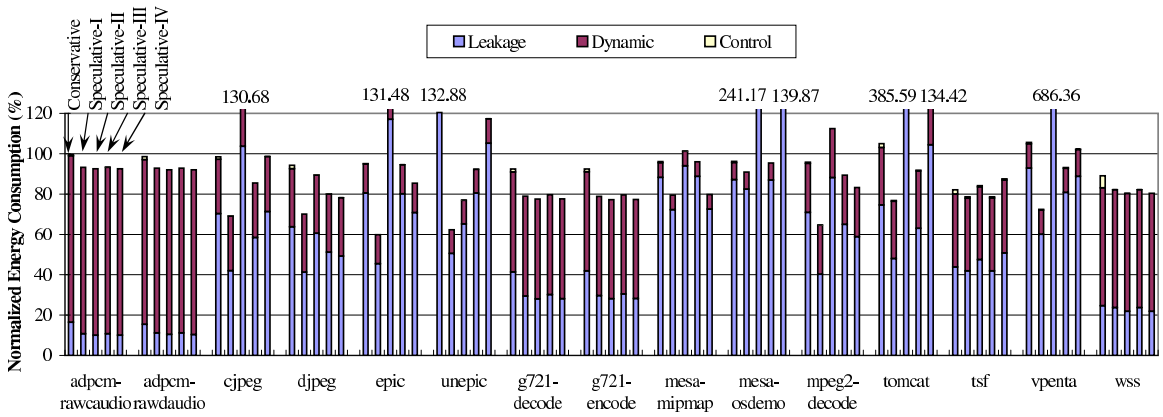

Figure 4. Normalized energy consumption of our optimization strategies.

in the state-destroying state for Speculative-II. Then, when the next access occurs, Speculative-I will incur 50 cycle delay (for reactivation), whereas Speculative-II 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, Speculative-I exhibits a better energy behavior than Speculative-II. There are, however, exceptions to this general trend: `adpcm-rawcaudio`, `adpcm-rawdaudio`, `g721-decode`, `g721-encode`, and `wss`. In these codes, the L1 replacement rate (the ratio between the number of L1 replacements and total L1 accesses) is very low (around 0.02) and L2 subblocks stay in energy-saving state longer (which works in favor of Speculative-II). In comparison, in some benchmarks (e.g., `epic`), the L1 replacement rate was over 80%, making the state-preserving strategy much more effective.

We now focus on Speculative-III and Speculative-IV. Recall that these two schemes differ from Speculative-I and Speculative-II in that they reactivate the L2 subblock when the corresponding L1 subblock is evicted. When data is moved from L2 to L1, Speculative-III places the corresponding subblock into the state-preserving state, whereas Speculative-IV puts it in the state-destroying mode. So, as far as a single subblock is concerned, Speculative-IV seems to be more energy-efficient. However, if all subblocks in a given L2 block are moved into L1, Speculative-IV 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 Speculative-III but fails in Speculative-IV. Consequently, in such cases, Speculative-III might perform better than Speculative-IV. The results in Figure 4 indicate that in six benchmarks Speculative-III consumes less energy than Speculative-IV (due to frequent memory accesses). In the remaining codes, Speculative-IV performs better than Speculative-III (due to lack of the above mentioned scenario).

When we compare Speculative-I and Speculative-III, we see that both of them preserve the data in L2, but Speculative-III 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 Speculative-I. However, its performance is better than Speculative-I in almost all cases. Finally, comparing Speculative-II and Speculative-IV, we observe that although both of them destroy data in L2, Speculative-IV has a better chance for avoiding main memory access, thanks to the early reactivation. In most of the benchmarks, Speculative-IV generated a better energy behavior than Speculative-II.

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 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 5 are the normalized energy-delay products (with respect to the original cache management without any leakage energy control). It is easy to see that Speculative-II and Speculative-IV do not perform well due to frequent main memory visits resulting from L2 misses. We observe, however, that the Speculative-

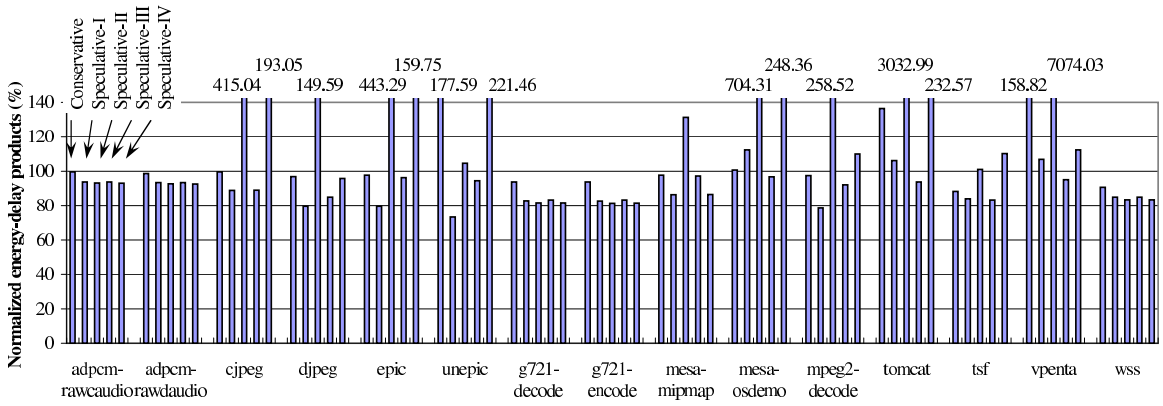

Figure 5. Normalized energy-delay products of our optimization strategies.

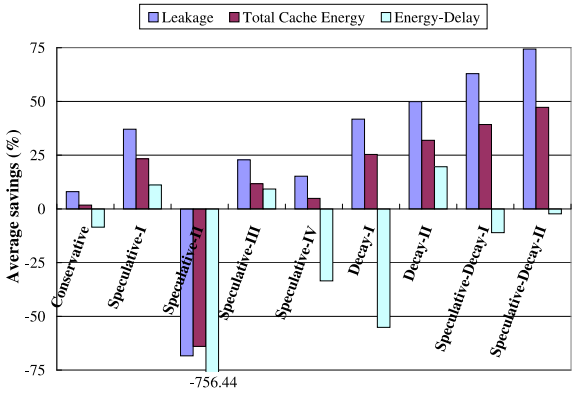

Figure 6. Average savings (over all benchmarks) for different optimization strategies.

I strategy reduces the energy-delay product by 11.1%, on the average. Apart from Speculative-I, only Speculative-III improves the energy delay product (9.3% on the average). State-destroying optimization strategies, on the other hand, increase energy-delay product by 8.5% (Conservative), 756.4% (Speculative-II), and 33.5% (Speculative-IV). 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 6 show the average values (percentage improvements) for our five optimizations across all benchmark programs for leakage energy, overall cache energy, and energy-delay product.

4. Comparison and Integration with Other Strategies

In [12], 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. In [12], they applied this strategy to L1 caches and showed that cache decay reduces the L1 cache leakage energy by a factor of four in Spec2000 applications without much impact on performance.

To compare this technique with our approach, we implemented cache decay in SimpleScalar [4] and performed experiments. The first implementation (called `Decay-I`) 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-destroying strategy in L1 and the state-preserving strategy in L2. In this second implementation (called `Decay-II`), the L2 cache is energy-managed 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-I). In both of these implementations, we used the threshold values used in [12] (i.e., 10K cycles for L1 and 1M cycles for L2). Note that it is not useful to apply the state-preserving leakage control in the L1 cache as the penalty for transitioning from the state-preserving leakage control state is larger than the latency to access L2 (50 cycles versus 10 cycles).

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

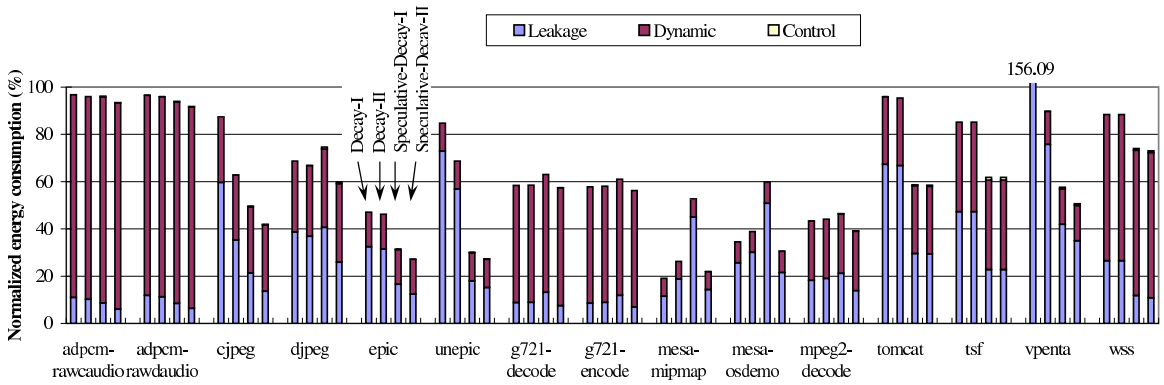

Figure 7. Normalized energy consumption of cache decay and combined strategies.

Speculative-I 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 7 and 8 show the normalized energy consumptions and energy-delay products, respectively, for these last four strategies mentioned above. It can be observed that Decay-I performs quite well and improves leakage energy consumption and overall cache energy by 41.8% and 25.3%, 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 55.1%. In most cases, Decay-II improves over Decay-I in both energy and energy-delay product. It improves leakage energy, total cache energy, and energy-delay product by 49.9%, 31.9%, and 19.6%, respectively. This is because in Decay-II, the L2 cache management does not destroy data, thereby preventing frequent main memory accesses. Also, as compared to Decay-I, it manages L2 leakage energy in subblock granularity. Recall that Speculative-I's leakage energy, total cache energy, and energy-delay product improvements were 37.1%, 23.3%, and 11.1%, respectively. So, Decay-I brings only a small improvement over Speculative-I in energy, but the former suffers from long execution times. Decay-II, however, performs better than Speculative-I in all aspects. It should also be stressed that while Speculative-I targets only the L2 cache, Decay-I and Decay-II target both caches and hence they have potentially larger optimization scope. In fact, comparing the savings *only* in the L2 cache shows that Speculative-I, Decay-I, and Decay-II reduce leakage energy consumption by 54.6%, 29.9%, and 39.37%.

Speculative-Decay-I outperforms Decay-II as far as energy consumption is concerned. It improves the unoptimized leakage energy by 62.9% and overall cache energy by 39.3%. However, it increases execution time and thus its energy-delay product (-11.1%) is worse than Decay-II. This is because of the following frequently-occurring scenario. When an L2 subblock is brought into L1, in Speculative-Delay-I, this subblock is placed into the state-preserving leakage control mode in L2. Later, when the block in L1 is evicted, the corresponding L2 subblock needs to be activated and the associated penalty is paid. In contrast, in Decay-II, under the same scenario, the same L2 subblock is not put in the state-preserving leakage control mode; therefore, it can be accessed quickly. However, in some benchmarks such as `epic`, before this scenario takes place, the entire L2 cache block can be evicted by Decay-II. In such cases, Speculative-Decay-I outperforms Decay-II.

Finally, Speculative-Decay-II generates the best energy results among these optimization strategies. As compared to the unoptimized case, it improves leakage and overall cache energy by 74.4% and 47.3%, respectively. These energy benefits are due to its aggressive optimization strategy in L2. More specifically, the two different methods (Speculative-I and cache decay) compete with each other to optimize L2 energy. Its energy-delay product, however, is -2.3%, which is worse than that of Decay-II (19.6%). Again, this is because of the scenario mentioned in the previous paragraph. The last four groups of bars in Figure 6 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. Conclusions

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

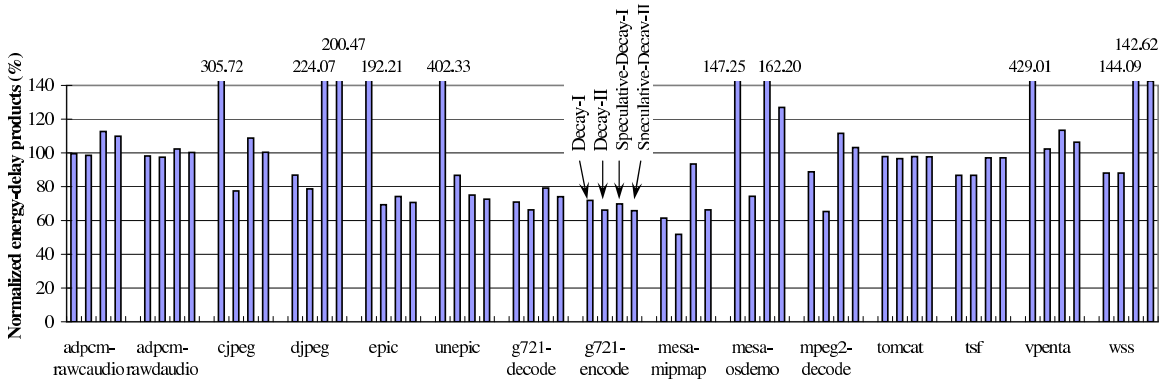

Figure 8. Normalized energy-delay products of cache decay and combined strategies.

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 (Speculative-I) 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. 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.

References

[1] Berkeley predictive model http://www-device.eecs.berkeley.edu.

[2] D. H. Albonesi. Selective cache ways: on demand cache resource allocation. In *MICRO-32*, pp. 248–259, Nov. 1999.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA-27*, June 2000.

[4] D. C. Burger and T. M. Austin. The SimpleScalar tool-set, Version 2.0. Technical Report 1342, Dept. of Computer Science, UW, June 1997.

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

[6] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko. Tuning garbage collection in an embedded java environment. In *HPCA-8*, Feb. 2002.

[7] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *ISCA-29*, May 2002.

[8] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *ISCA-29*, May 2002.

[9] K. Itoh, K. Sasaki, and Y. Nakagome. Trends in low-power RAM circuit technologies. *Proceedings of IEEE*, 83(4):524–543, Apr. 1995.

[10] N. Jouppi and S. Wilton. Tradeoffs in two-level on-chip caching. In *ISCA-21*, pp. 34–45. IEEE Computer Society Press, 1994.

[11] H. Kawaguchi, K. Nose, and T. Sakurai. A super cut-off CMOS scheme for 0.5V supply voltage with pico-ampere standby current. *Journal of Solid-State Circuits*, 35(10):1498–1501, Oct. 2000.

[12] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *ISCA-28*, June 2001.

[13] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *MICRO-30*, pp. 330–335, Dec. 1997.

[14] T. May and M. Woods. Alpha-particled-induced soft errors in dynamic memories. *IEEE Trans. on Electron Devices*, ED-26(1), Jan. 1979.

[15] 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.

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

[17] 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.

[18] L. Villa, M. Zhang, and K. Asanovic. Dynamic zero compression for cache energy reduction. In *MICRO-33*, Dec. 2000.

[19] 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 *HPCA-7*, Jan. 2001.

[20] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: A static-power-efficient cache design. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, Sept. 2001.