Enhancing the L1 Data Cache Design to Mitigate HCI

Alejandro Valero*, Negar Miralaei†, Salvador Petit*, Julio Sahuquillo*, and Timothy M. Jones†
*Department of Computer Engineering, Universitat Politècnica de València, Spain, *alvabre@gap.upv.es,* {*spetit,jsahuqui*}*@disca.upv.es*
†Computer Laboratory, University of Cambridge, UK, {*negar.miralaei,timothy.jones*}*@cl.cam.ac.uk*

Abstract—Over the lifetime of a microprocessor, the Hot Carrier Injection (HCI) phenomenon degrades the threshold voltage, which causes slower transistor switching and eventually results in timing violations and faulty operation. This effect appears when the memory cell contents flip from logic '0' to '1' and vice versa. In caches, the majority of cell flips are concentrated into only a few of the total memory cells that make up each data word. In addition, other researchers have noted that zero is the most commonly-stored data value in a cache, and have taken advantage of this behavior to propose data compression and power reduction techniques. Contrary to these works, we use this information to extend the lifetime of the caches by introducing two microarchitectural techniques that spread and reduce the number of flips across the first-level (L1) data cache cells. Experimental results show that, compared to the conventional approach, the proposed mechanisms reduce the highest cell flip peak up to 65.8%, whereas the threshold voltage degradation savings range from 32.0% to 79.9% depending on the application.

Index Terms—Cache memories, cell flip peaks, Hot Carrier Injection, threshold voltage degradation.

1 Introduction

Modern day computer systems have benefited from being designed and manufactured using an ever-increasing budget of transistors on very reliable integrated circuits. However, as technology moves forward, such a "free lunch" is over as increasingly smaller technology nodes pose significant reliability challenges. Not only variations in the manufacturing process make the resulting transistors unreliable at low voltage operation, but they take less and less time to wear out, decreasing their lifetimes (from tens of years in current systems to 1-2 years or fewer in the near future) and making them more prone to failures in the field. Thus, lifetime reliability must be treated as a major design constraint. This concern holds for all kind of computing devices ranging from server processors to embedded systems like tablets and mobiles, where lifetime is an assertive requirement and the market share strongly depends on their reliability.

The two main phenomena that speed up aging are referred to as Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI). The former effect increases with transistor activity over the lifetime of the processor; that is, when the cell contents flip from logic '0' to '1' and vice versa, leading to threshold voltage (V_{th}) degradation ($\Delta V_{th} \propto \sqrt{active\_time \times num\_flips / num\_cycles}$) [8], which in turn causes an increase in transistor switching delay ($\Delta Delay \propto 1/(V_{dd} - \Delta V_{th})^{\alpha}, \alpha > 1$) [8] and can result in faulty operation. On the other hand, BTI accelerates the cell degradation when a given logic value is stored for a long time. Other parameters that speed up aging are the high temperature and the low supply voltage (V_{dd}). Overall, HCI is accentuated in those microprocessor components where data are more frequently written, like first-level (L1) caches and register files. Moreover, these memory components are implemented as arrays of 6-transitor SRAM cells, and, given that wearout affects any cell transistor, the performance and availability of a whole array of cells (e.g., a cache line) can be affected.

Prior research work has already analyzed cache degradation mainly due to BTI effects. In [1], BTI is diminished by periodically inverting the stored logic value in the cells. Gunadi *et al.* [2] periodically write normal and complemented data in caches and balance the accesses across the cache sets by using an LFSR to seed the hash set index function. Shin *et al.* [5] implement redundancy (additional cell regions) in the cache, which are used when the system identifies that the original regions are being affected by aging. Other work [6] modified the original SRAM cell design used to implement the issue queue. Finally, in [8], Tiwari and Torrellas dynamically adapt the effects of temperature and voltage on aging to enlarge the processor lifetime. However, to the best of our knowledge, there is no previous work at architectural level addressing HCI aging in caches.

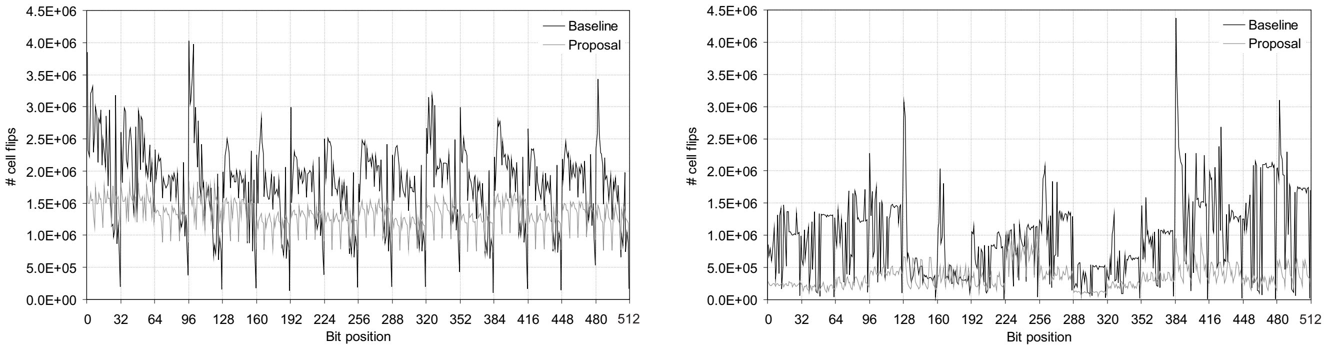
This work focuses on enlarging the cache lifetime by reducing the V_{th} degradation or simply dV_{th}. More precisely, we provide a homogeneous degradation of the different cache cells belonging to the same cache line through two main contributions. First, we characterize the cell flips that applications cause to each specific memory cell. We find that most applications exhibit regular flip patterns, although flips are not always uniformly distributed, instead being concentrated on a small number of bits within the 512-bit cache lines. Results also confirm previous work [9] [4] claiming that most applications write a significant number of *near-zero* and zero data values into the cache. This behavior has been exploited in the past to address static energy consumption [9] and performance with data compression [4].

Based on the previous observations, we devise an HCI-aware mechanism aimed at reducing HCI effects. The proposal pursues two-fold objectives: i) to spread the bit flips evenly across the memory cells and ii) to reduce the overall number of bit flips. Results show that, compared to a conventional cache design, the highest flip peak saving for the whole cache can be as high as 65.8%, whereas the dV_{th} reduction ranges from 32.0% to 79.9% depending on the application.

2 Characterization Study

We have characterized the bit flip pattern across the SPEC2006 benchmark suite. For illustration purposes, we use an integer (*perlbench*) and a floating-point (*sphinx3*) benchmark since applications of these types use data with different internal representations. Figure 1 depicts the number of bit flips encountered in the memory cells of a 16KB 4-way L1 data cache for both applications under the baseline approach (i.e., no flip mitigation is employed) and under our proposed mechanism. Please refer to Section 4 for simulation details. For each of the 512-bit positions in a 64B cache line, the figure plots the sum of the number of bit flips in every cache line. Note that the written data follow the little-endian representation.

For the baseline scheme, the *perlbench* integer benchmark shows a regular flip pattern across the sixteen 32-bit words cache lines. Peaks concentrate on bits within the Least Significant Byte (LSB) of the words, whereas bits in the MSB account for a much lower



a) *perlbench*

b) *sphinx3*

Fig. 1. Number of flips on each bit position of the cache lines for the baseline approach and our proposed scheme.

number of cell flips. Such a behavior is quite common for the remaining SPEC2006 integer applications. One of the main reasons for this is that processors store a significant amount of near-zero or narrow values in caches [4]. These values are used to index arrays and matrices, which are extensively accessed inside program loops. Another important fact is the over-provisioning; that is, programmers usually define relatively large data types (e.g., 4-byte integers) for storing a small value, which is normally represented with a few bits distinct from zero.

In contrast, the *sphinx3* floating-point application shows a non-uniform flip pattern due to the IEEE-754 data representation. Nevertheless, even in floating-point benchmarks, some peaks for the baseline can be identified in the LSB of some words (e.g., those starting at bits 128 and 384 of the cache lines in the example).

3 THE BW APPROACH

3.1 Basic Mechanism: Shifting Bytes within Words

The previous study has shown that the memory cells containing the LSB of a word experience bit flip peaks across all the cells in a cache line, especially for integer applications. These LSB cells age the most from HCI because of the relationship between HCI and flip activity. To mitigate HCI wearout, our initial scheme implements a rotation shift mechanism that distributes the flips located on the peaks across several bit positions within the words. In particular, the proposed technique, called BW, periodically shifts the bytes within the words on a round-robin basis, which implies 4 *x-size* shift functions for 4-byte words where *x* means the number of bytes to be shifted. For instance, a 1-size shift for a given word writes the LSB into its next byte (i.e., those cells that originally hold the second LSB) and so on; the original MSB is written into those cells originally holding the LSB.

For hardware simplification purposes, all the 16 words in the 64-byte cache line follow the same shift function at any given time. Thus, each cache line simply requires one pair of control bits to keep track of the current shift. When the line is read, these bits are used to realign the word bytes and forward them in their correct positions to the processor.

An important design issue is the duration in time of a given shift, that is, for how long a shift function is maintained before applying the subsequent shift. For this purpose, the execution time is divided into phases. A shift transition occurs when a cache line is written for the first time within a phase. This is detected by adding a single control bit per cache line to indicate whether the line has been ever written in the phase or not. Of course, this bit is reset every time a new phase starts. The shift function keeps valid for all the subsequent cache accesses performed during the remaining of the current phase, as well as for the subsequent phase/s until a write operation rises.

Notice that such a behavior implies that, at a given time, the different cache lines can follow distinct shift functions since writes do not act in a synchronized behavior. Moreover, a small number of the cache lines are usually being accessed at a given point in time.

We found in our simulation environment that a phase length of 8M processor cycles noticeably reduces the high flip peaks. Very large phases result in less shifts and longer shift functions, which could not significantly reduce those high peaks, whereas very short phases could largely increase the overall number of flips, leading to high peaks.

TABLE 1
Application characterization according to the percentage of zero bytes.

20%-40%	40%-60%	60%-80%
soplex, perlbench	*tonto, sphinx3, mcf*	*astar, h264ref, dealII*

3.2 BW Mechanism Enhancement

There are several reasons that explain why current applications write a significant number of zeros. First, memory is usually initialized to zero when it is allocated. Second, false boolean values and NULL pointers are represented with zero, as well as most data in sparse matrices. To enhance the mechanism we quantified the percentage of bytes written to zero in the L1 data cache. Table 1 summarizes the results for the 8 SPEC2006 applications (4 integer and 4 floating-point) that experience the highest number of flips. As observed, all of these applications write zero bytes more than 20% of the time. Moreover, in three of them, this is more than 60% of the time. Based on these results, skipping zero byte values from being written into the cache would reduce the number of flips.

To leverage this fact, the BW technique is enhanced with the Non-write Zero Byte (NZB) mechanism. NZB works as follows. On a cache miss or a write hit, each byte to be written into the cache is compared to zero. If the comparison matches, a control bit is set to indicate that it represents a zero, but the cache byte value is not modified. On a cache read hit, the corresponding control bits associated with the bytes in the target line are checked, and if any of them is set, a zero byte is forwarded to the processor instead of the byte stored in the cache. Notice that simple hardware is required to compute the control bits and forward zero bytes [9].

To help understanding of the positive effects of the proposed NZB scheme, Figure 2 shows an example of three consecutive write operations (W_i) to the same cache line consisting of 2 bytes. The first write updates the cache line with the data to be written and both control bits are reset. Then, write W_2 does not update the line since both bytes to be written are zero. Instead, both control bits are set and the line contents remain untouched. Finally, W_3 writes a couple of bytes distinct from zero and both control bits are reset. The tick and cross symbols over the bits in the line show the effectiveness of NZB. The former refers to those bits where NZB saves a cell flip with respect to the baseline, while the latter is located in those bits where NZB introduces an additional flip. Note that those flips located in the

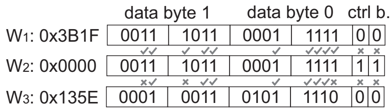


Fig. 2. Example of NZB with three consecutive writes to the same line.

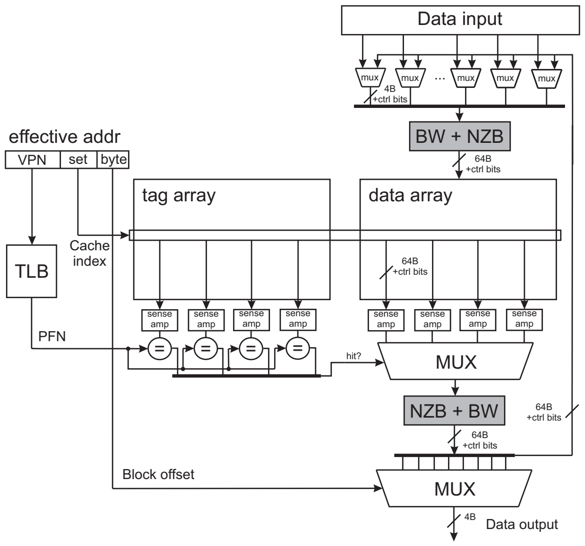


Fig. 3. Block diagram of the L1 data cache access including the proposed components (grey boxes).

control bits have also been considered. The overall effect is to save 17 flips, but incur an additional 7, meaning a net reduction of 10 flips.

Finally, notice that both BW and NZB would not speed up BTI wearout since these techniques reduce the amount of time that logic '0' is stored in the cell, which is the main cause of BTI wearout [1].

3.3 Hardware Implementation, Overhead, and Timing

Figure 3 depicts a cache block diagram with the two proposed mechanisms represented as grey boxes. On a cache read hit, after the way multiplexer selects the target line from the selected set, its contents and the associated control bits are forwarded to the NZB read circuit. Once the NZB read circuit has forwarded the zero bytes, the BW read circuit realigns the bytes and serves the line to the processor.

On a cache write hit, the contents stored in the target line are read and forwarded to the upper multiplexers, which compose the line to be stored jointly with the input data. After that, the BW write circuit rotates the bytes of each word in the line according to the corresponding shift function. Then the NZB write circuit computes the NZB control bits and prevents the zero bytes from being written into the data array. Note that on a cache miss, the same circuitry is used to store the incoming data.

Remember that BW requires 3 control bits per cache line: 2 bits to record the shift function and a single bit to discern whether the shift function must change on the next write to the line. In addition, the NZB technique requires 1 control bit per byte, which results in an overhead of 16K bits for the studied 16KB L1 data cache. Overall, the storage overhead required by both schemes together is by 12.7% of the cache data array capacity.

To study the impact on the cache access time, we considered the involved delays. The CACTI 5.3 tool [7] reports a 0.088ns delay for each 4 to 1 multiplexer used to implement the shift functions, and a 0.765ns L1 cache access time. Considering the largest path, which in our proposed architecture is given by the write operation, the access time becomes 0.941ns. For the assumed 3GHz processor, therefore, the difference (in ns) over the original delay is masked when the access time is quantified in processor cycles. That is, the additional circuitry has no impact on the data array access time (in cycles), although it could impact on other processor designs (e.g., those working at a higher frequency). In such a case, an optimized or alternative design of the additional circuitry would be required. However, the design of this circuitry is beyond the scope of this paper.

TABLE 2
Architectural machine parameters.

Issue policy	Out of order
Branch predictor type	Hybrid gShare + bimodal
Branch predictor penalty	10 cycles
Fetch, issue, commit width	4 instructions/cycle
ROB size (entries)	256
# Int/FP ALUs	4/4
L1 data & insn caches	16KB, 4-way, 64B-line, 1-cycle tag array, 3-cycle data array
L2 unified cache	256KB, 8-way, 64B-line, 6-cycle tag array, 10-cycle data array
L3 unified cache	4MB, 16-way, 64B-line, 11-cycle tag array, 23-cycle data array
Main Memory	200-cycle

4 Experimental Results

An extended version of the Multi2Sim simulation framework [10] has been used to implement both NZB and BW approaches. Experiments were performed for the 32-bit x86 ISA with the *ref* input set, while results were collected simulating 500M instructions after skipping the initial 500M instructions. Table 2 summarizes the main architectural parameters. All the cache access times were obtained from CACTI for a 3GHz processor speed and a 32nm technology node.

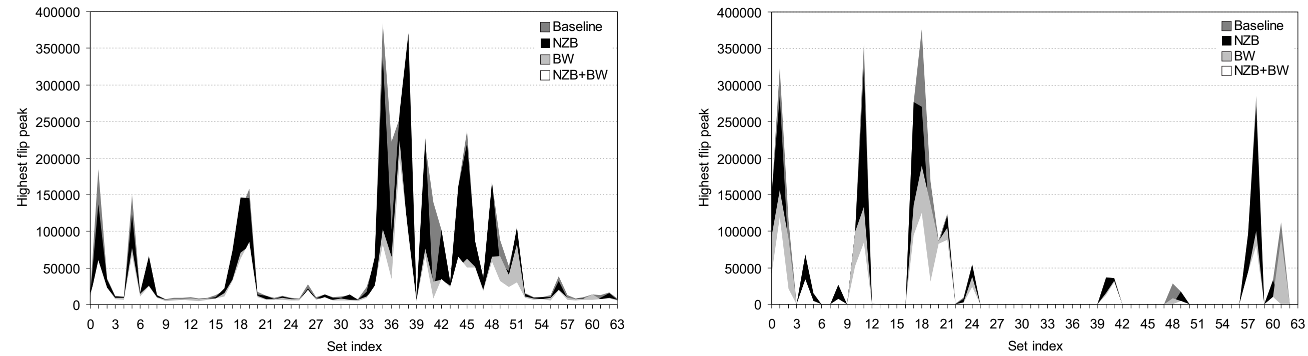
4.1 Cell Flip Reduction Analysis

This section identifies those cache sets where applications induce a significant number of cell flips and quantifies the number of flips the proposed mechanisms save. Results are shown for the baseline scheme, NZB and BW working alone, and both schemes working together (NZB+BW).

Figure 4 plots the raw number of flips for the memory cell within each cache set that holds the highest flip peak for *perlbench* and *sphinx3*. As observed, the highest peak value widely varies across the cache sets due to the non-uniform distribution of accesses across them. The enhanced NZB+BW scheme is the one that most minimizes the number of flips, followed by BW and NZB working in isolation. For *perlbench*, the cache set with index 35 accumulates the highest cache flip peak for the baseline approach, whereas such a peak is located in set 37 for the NZB+BW, allowing a highest flip peak reduction by 48.2% for the entire cache. On the other hand, the highest cache flip peak for *sphinx3* is always located in set 18 regardless of the studied mechanism. In this case, the NZB+BW highest flip peak saving is up to 66.7% over the baseline.

Figure 5 summarizes the highest cache flip peak for all the analyzed benchmarks, confirming that combining both NZB and BW allows the maximum flip reduction. BW turns to be much more effective than NZB, since the BW savings range from 5.9% (*soplex*) to 65.4% (*astar*), whereas these percentages for NZB range just from 0.1% (*tonto*) to 18.0% (*h264ref*). Overall, applying them jointly leads to flip savings from 25.0% (*dealII*) to 66.7% (*sphinx3*).

The figure also shows the highest flip peak in the additional control bits used by NZB+BW, represented with a cross symbol. For all the studied benchmarks, such a peak rises in the NZB control bits since their activity is higher than in those of the BW mechanism. Nevertheless, for most applications (6 out of 8) the data array has a higher flip peak than in the NZB control bits. This is due to flips in the NZB control bits are only occuring from byte write sequences in the same byte location with a non-zero value followed by a zero value and vice versa, whereas all sequences with a non-zero value followed by a distinct non-zero value induce cell flips in the data array. For *h264ref*,



a) *perlbench* b) *sphinx3*

Fig. 4. Highest flip peak on each cache set for the baseline and proposed approaches.

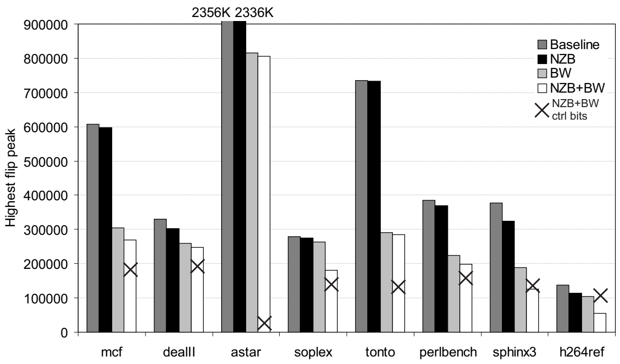


Fig. 5. Maximum cache flip peak for all the studied applications.

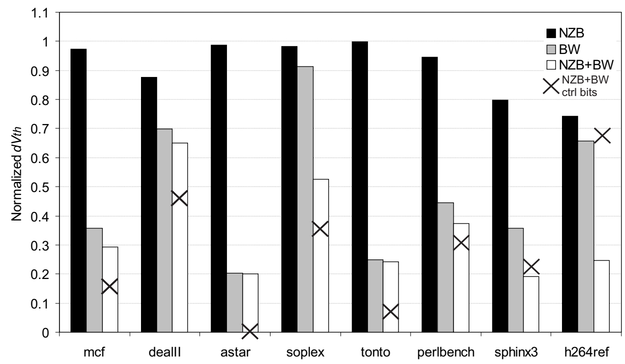


Fig. 6. Normalized dV_{th} for the memory cell with the highest flip.

the peak in the control bits is relatively high because this application writes a large number of zero bytes (see Table 1) and most sequences are those affecting the NZB control bits. Still, the peak in the control bits is always much lower than that of the baseline's data array for all the studied applications. Overall, the flip savings range from 22.7% (*h264ref*) to 65.8% (*astar*).

Recall that the flip savings of NZB+BW are also reported in Figure 1, where can be appreciated that the proposal greatly reduces the flip peaks and equalizes out flips across the bytes within each line.

4.2 Analysis of dV_{th} Savings

This section analyzes the V_{th} degradation caused solely by the HCI phenomenon. Results were gathered just for the memory cell with the highest number of flips (not necessarily the same cell across the studied approaches), which is the one that suffers the highest HCI wearout. Results have been computed using a standard dV_{th} formula [8] and assuming a 3-year lifetime for our 32nm technology node [3]. To complete this execution period, we have assumed that the simulated amount of 500M instructions is repeated over and over until the established period is reached.

Figure 6 depicts the normalized dV_{th} of the proposed approaches with respect to the baseline. The V_{th} degradation on the NZB+BW control bits is also plotted. Results confirm that dV_{th} is highly related to the number of flips, since similar to the previous analysis, *h264ref* and *astar* are those applications that less (by 32.0%) and most (by 79.9%) reduce dV_{th}, respectively.

5 Conclusions

This paper has shown that the proposed rotation shift and non-write zero techniques are effective microarchitectural mechanisms to mitigate wearout in L1 data caches due to HCI effects. Results showed that the highest flip of the entire cache can be reduced up to 65.8%, while the largest dV_{th} reduction is up to 79.9%.

As for future work we plan to evaluate the mechanisms on SMT processors where flip and dV_{th} savings would be even higher given that the number of flips could exacerbate due to thread interferences.

Acknowledgments

This work has been supported by the Spanish *Ministerio de Economía y Competitividad* (MINECO), by FEDER funds through Grant TIN2012-38341-C04-01, by the Intel Early Career Faculty Honor Program Award, by a HiPEAC Collaboration Grant funded by the FP7 HiPEAC Network of Excellence under grant agreement 287759, and by the Engineering and Physical Sciences Research Council (EPSRC) through Grants EP/K026399/1 and EP/J016284/1. Additional data related to this publication is available in the data repository at https://www.repository.cam.ac.uk/handle/1810/249006.

References

[1] J. Abella *et al.*, "Penelope: The NBTI-Aware Processor," in *Proceedings of MICRO-40*, 2007.
[2] E. Gunadi *et al.*, "Combating Aging with the Colt Duty Cycle Equalizer," in *Proceedings of MICRO-43*, 2010.
[3] E. Mintarno *et al.*, "Workload-Dependent NBTI and PBTI Analysis for a sub-45nm Commercial Microprocessor," in *Proceedings of IRPS*, 2013.
[4] G. Pekhimenko *et al.*, "Base-delta-immediate Compression: Practical Data Compression for On-chip Caches," in *Proceedings of PACT-21*, 2012.
[5] J. Shin *et al.*, "A Proactive Wearout Recovery Approach for Exploiting Microarchitectural Redundancy to Extend Cache SRAM Lifetime," in *Proceedings of ISCA-35*, 2008.
[6] T. Siddiqua and S. Gurumurthi, "Recovery Boosting: A Technique to Enhance NBTI Recovery in SRAM Arrays," in *Proceedings of VLSI*, 2010.
[7] S. Thoziyoor *et al.*, "CACTI 5.1," *HP Development Company, Palo Alto, CA, USA. Tech. Report*, 2008.
[8] A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," in *Proceedings of MICRO-41*, 2008.
[9] R. Ubal *et al.*, "Leakage Current Reduction in Data Caches on Embedded Systems," in *Proceedings of PerCom*, 2007.
[10] ——, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," in *Proceedings of SBAC-PAD-19*, 2007.