Application-Specific Memory Partitioning for Joint Energy and Lifetime Optimization

†Haroon Mahmood, †Massimo Poncino, ‡Mirko Loghi, †Enrico Macii
†Politecnico di Torino, Torino, 10129, ITALY
‡University di Udine, Udine, 33100, ITALY

Abstract—**Power management of caches based on turning idle cache lines into a low-energy state is also beneficial for the aging effects caused by Negative Bias Temperature Instability (NBTI), provided that idleness is correctly exploited; unlike energy, aging, being a measure of delay, is in fact a worst-case metric.**
In this work we propose an application-specific partitioned cache architecture in which a cache is organized as a set of independently addressable sub-blocks; by properly using the idleness of the various banks to drive how the partition is determined, it is possible to extend the effective lifetime of the cache while saving extra energy.
Two are the distinctive features of our approach: First, we allow the cache sub-blocks age at different rates, achieving a sort of graceful degradation of performance while extending lifetime beyond the limits of previously published works. Proper architectural arrangements are also introduced in order to cope with the issue of using a progressively smaller cache. Second, the sub-blocks have non-uniform sizes, so to maximally exploit idleness for joint energy and aging optimization.
Simulation results show that it is possible to extend the effective lifetime of the cache by more than 2x with respect to previous methods, while concurrently improving energy consumption by about 50%.

I. Introduction

Aging of devices has emerged as the latest challenge brought by technology scaling. Thinner oxide layers, higher electric fields and operating temperatures, induce adverse physical and chemical phenomena that cause transistors to deteriorate their performance over time. The three main sources of device aging are Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), and Time Dependent Dielectric Breakdown (TDDB). Of the three, BTI, and in particular Negative Bias Temperature Instability (NBTI) appears to be the most critical one in sub-65nm technologies [1]. NBTI affects PMOS devices under negative bias (i.e., when a logic "0" is applied to the gate terminal), resulting in a temporal drift of the threshold voltage, which translates into a delay increase over time. However, NBTI is partially reversible if a logic "1" is applied to the gate terminal [1].
A large amount of research has recently been published addressing NBTI aging from the design and EDA perspective, by trying to act on design variables that regulate the aging process [2]—[6]. SRAM memories, in particular, have received special attention: One reason is their criticality in the determination of the overall system performance; another and more subtle motivation is due to the fact that NBTI value dependence is weaker in memory cells: due to its symmetric structure, a SRAM cell ages regardless of the value stored.
The most effective solutions rely on the observation that typical power management strategies (i.e., voltage scaling for dynamic power and power/ground gating for static power) can be exploited to reduce NBTI-induced aging [7], [8]. Therefore, proper revisitation of power-managed memory/cache architectures according to an aging-related metric can achieve concurrent energy and aging improvements [16], [15], [17].
Our work is focused on caches and is close in scope to that of [17], in which a multi-bank cache implementation with an improved aging profile was proposed: the work leverages the idea introduced in [16], that is, the use of time-varying cache indexing strategy (called *dynamic indexing*) to achieve perfectly uniform distribution of idleness over the cache lines. The work of [17] extends this paradigm to a multi-bank cache architecture in which dynamic indexing is applied to individual banks rather than cache lines to achieve concurrent (static) energy (thanks to the cache partitioning) and aging benefits. In practice, [17] implements a coarse-grain version of [16].
Regardless of the granularity, the two methods share two features: first, dynamic indexing causes all the power management units (cache lines or cache blocks) to age identically. Second, all power management units have same size.
In this work we explore a partitioned cache architecture in which we relax both constraints. First, we allow the **different blocks into which the cache is partitioned to age at different rates**. This implies that some cache block will become unreliable first, and the cache will keep on functioning with a reduced efficiency (or equivalently, with a progressively smaller cache). Another implication of this graceful aging is that a proper aging metric is required for a fair comparison against the method of [17]. To this purpose, we introduce the concept of *Effective LifeTime (ELT)*, that is, the product of lifetime and size of a memory block. ELT conceptually measures *how much memory can be used for how much time*.
Secondly, we allow the **blocks into which the cache is partitioned to have different sizes**. In order to simplify implementation and demonstrate a first proof-of-concept, in this work we only consider the case of two partitions. Since the sizes of the two sub-blocks become an optimization

This work is supported in part by the European Commission under grant n. 247999 "COMPLEX": COdesign and power Management in PLatform-based design space EXploration.

variable, we propose two partitioning algorithms that combine in different ways ELT-driven partitioning with the swapping of a small subset of cache indices.

As results will show, relaxing these constraints provides better results in the (leakage, ELT) space. In particular, we improve ELT by over a factor of 2 for a monolithic cache, over the approach of [17], while concurrently reducing total energy consumption by about 50%, on average.

II. Background and Related Work

A. Background

In this section we summarize the basic issues related to NBTI-induced aging in SRAM cells; for a more in-depth analysis of NBTI mechanisms we refer the reader to the introductory papers in the literature (e.g., [1]).

The threshold voltage drift induced by NBTI does not truly affect the *delay* of a SRAM cell. Rather, it impacts its stability, since it alters its Static Noise Margin (SNM), defined as the minimum DC noise voltage required to change the state of the cell. When the SNM of a cell falls below a threshold that allows safe storage of data it cannot be safely read or written.

Another peculiarity of NBTI effects on SRAM cells is that the value dependence is quite weak: the symmetry of the cell causes it to age whatever the value it stores. The best-case degradation occurs when both inverters in the cell exhibit the same amount of degradation, that is, when a cell stores a 0 and a 1 with equal probability [9].

B. Related Work

Solutions proposed to mitigate NBTI effects in SRAMs fall in three main categories. One class includes methods that try to equalize cell value probabilities [9], [10]. In [9] the authors provide hardware and software schemes to periodically invert the entire content of a memory so as to guarantee a perfectly balanced probability. A similar strategy was pursued by [10], yet at a word granularity and with a much shorter inversion frequency (thousands of cycles).

Another type of approach aims at designing customized NBTI-resilient cells [11], [12]. In [11] a new cell structure is proposed consisting of a set of NAND gates arranged so that minimum degradation ratio for all PMOS transistors in the cell is obtained. Another solution called *recovery boosting* [12] allows both PMOS devices in the cell to be put into the recovery mode by raising the ground voltage and bitlines to the nominal voltage through modification of each memory cell.

A third class of solutions is based on the exploitation of the aging benefits provided by low-energy states [15], [16], [17]. Assessment at the architectural level on entire memory blocks of power management solutions (based on both DVS and power gating) were evaluated in [15].

The work of [16] introduces a *dynamic indexing* scheme in which the cache indexing function is modified over time in order to achieve a uniform distribution of idleness over the cache lines; in this way all the leakage saving opportunities can also be used for aging reduction. The work of [17] can be viewed as a coarse-grain extension of [16], and implements a uniform-size, multi-bank cache with the purpose of achieving a better design point in aging/energy design space.

III. Motivation and Concept

Consider a power-managed cache in which individual lines can be turned into a low-power state based on their access pattern (e.g., [13], [14]); if we now simulate a given workload we can extract the *exploitable idleness* (i.e., idle intervals longer than some breakeven time) of each line.

Given that idleness can be exploited for both energy and aging reduction, it is straightforward to observe that different characteristics of the idleness profile matter for the two metrics. For energy, it is clearly the average value that matters: energy savings for each line will cumulate proportionally to the idleness of the line. For aging, conversely, it is the worst case that matters: the line that becomes unusable first will cause the entire cache to fail. Since due to the very principle of locality, the distribution of idleness will in general not be uniform, average and worst case will differ.

These considerations have inspired the works of [16] and [17]; using different architectural arrangements, both solutions aim at the same objective, that is, having individual cache lines ([16]) or cache blocks ([17]) to "die" at the same time. This corresponds to making *average and worst case to coincide*.

A different strategy is indeed possible, namely, to allow the different blocks to **age at different rates**. This implies that some cache block will become unreliable first, and the cache will keep on functioning with a reduced efficiency (or equivalently, with a progressively smaller cache). Implementation of such a strategy requires the definition of at least two issues: the management of "dead" blocks and the evaluation of lifetime.

A. Architectural Support

A first requirement is to have an aging sensor that is able to monitor the aging of the corresponding block. The implementation of [18] perfectly fits the purpose because it can be easily embedded into an existing memory array and it is extremely compact. Since we need a monitor for each block, the proposed strategy is more suitable for a coarse-grain implementation; even if the monitor of [18] can easily be added to each line (it is essentially a modified SRAM cell), overhead consideration suggest that a larger block should be used as unit of power/aging management. Therefore, we compare our method to that of [17], in which *sets of contiguous cache lines* were used for power/aging management.

Assuming therefore a smaller number of blocks into which the cache can be partitioned, a second architectural issue concerns how to manage "dead" cache blocks. Two options are possible. In one scenario, after a block dies we can view the cache as becoming progressively smaller, similar to what is done in the DRI cache proposed in [13]. Although possible, this implies the re-design of the cache indexing mechanism (a smaller cache results into extra index bits, depending on how small it becomes); furthermore, it would limit the size of the block to manageable sizes (proper powers of two). Therefore, we adopt another scenario with simpler management. As soon as one block dies, we simply force the corresponding lines to become indefinitely invalid, and disable any kind of replacement for those lines: any further access to these lines will result on a miss. It is clear that this has impact on performance, but will allow to use the cache longer than without any aging management; in other terms, the performance of the cache will be *identical to the original one until the time the latter will die*, then it will become inferior.

B. Metrics

Another implication of this graceful aging is that a proper aging metric is required for a fair comparison against previous works. To this purpose, we introduce the concept of *Effective LifeTime (ELT)*, that is, the *product of lifetime and size of a memory block*. ELT conceptually measures *for how much time a memory block of a given size can be used.* Figure 1 pictorially describes the concept of ELT.

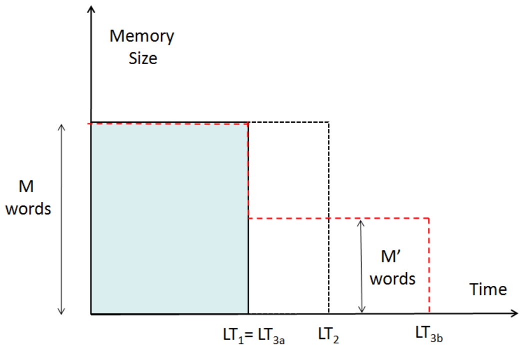

Fig. 1. Concept of Effective Lifetime.

The solid line enclosing the filled area denotes the aging profile of a regular memory: M words are usable reliably for an amount of time equal to LT_1. By using solutions such as [16], [17] we can extend the lifetime upto LT_2, and the memory still becomes unusable as a whole (dotted line). With our approach (dashed line), we will use the entire memory until LT_{3a} equal to original lifetime, but then disable the earliest failing block so that we can still use a smaller (say, $M' < M$ words) memory, yet for a longer time (up to LT_{3b}). The figure shows a simplified case in which only the cache is partitioned into two blocks.

The ELT is the area below the various aging profiles. Thus, the rationale is that, depending on the idleness distribution and on how we partition the memory, it is possible that $ELT_3 = M \cdot LT_{3a} + M' \cdot (LT_{3b} - LT_{3a}) > ELT_2 = M \cdot LT_2$

IV. Aging-Driven Cache Partitioning

A. Architecture

We assume a direct-mapped cache with $L = 2^n$ lines $(l_0, \ldots, l_{L-1})$, where n is the number of the index bits of the cache address. We want to split the cache into M blocks $B_0, \ldots, B_{M-1}$, of sizes $S_0, \ldots, S_{M-1}$, addressed using $n_0, \ldots, n_{M-1}$ bits, respectively. We assume $M \leq 4$ in order to limit the hardware overhead required for address decoding and wiring. Figure 2 shows the conceptual architecture and the relevant quantities.

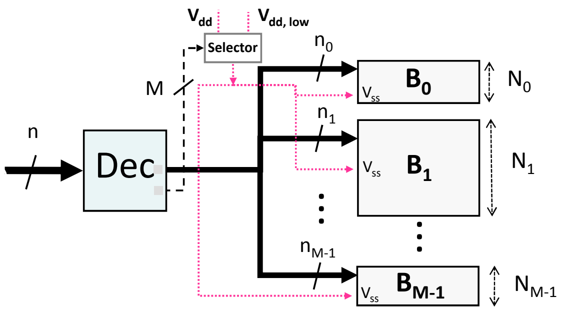

Fig. 2. Variable-Size Partitioned Cache Architecture.

The figures assumes the use of voltage scaling for implementing the low-energy states for the blocks (denoted by the dotted signal from the dual supply voltage selector). Voltage scaling is the only viable choice for the standard memory blocks provided by the memory compiler in our target technology. Moreover, voltage scaling allows to preserve the contents of the memory block in the standby state with a better energy/performance tradeoff [16].

The decoding block Dec in the figure serves two purposes: remapping the address on the proper block and asserting the standby signals for the M blocks.

B. Aging Model

The lifetime of a given block will be determined by the earliest failing line, which in turn is determined by the earliest failing cell. Besides the amount of idleness, also the value stored in the cell has a strong impact on its lifetime [9]. We have therefore accurately characterized using HSPICE the aging of a SRAM cell with respect to the percentage of idleness (exploited through voltage scaling) and for the two extreme cases of stored values (a fixed 0 or 1, and a 50% probability of storing a 0 or 1 – respectively worst and best case). Figure 3 shows the two curves, which are used by our algorithms to translate idleness into aging. Lifetime has been defined as the time after which the SNM has decreased by more than 20%. The curves are not linear, but there is a clear correlation between the two quantities.

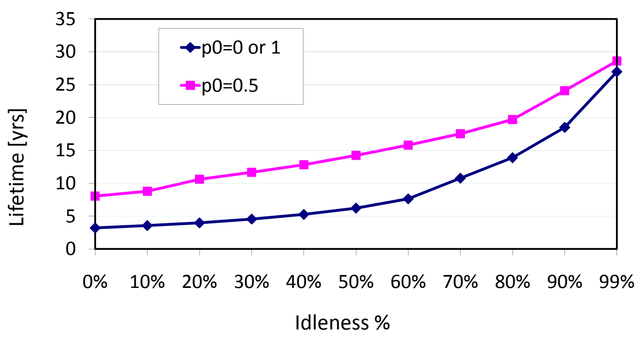

Fig. 3. Lifetime vs. Idleness.

C. Lifetime Optimization

The problem we are trying to solve can be stated as follows: *Given an idleness profile $\mathbf{I} = \{i_1, \ldots, i_L\}$ for a cache of L lines, and a number of banks M, generate a profile-specific partition of the cache into M banks $\mathbf{B} = B_0, \ldots, B_{M-1}$ so that the ELT of the partitioned cache is maximized.*

The problem can be solved by observing, as discussed in Section IV-B, that the lifetime of a block is determined by the line with the least idleness. Using the relation between actual lifetime and idleness of Figure 3, we can derive an analytical formula for the ELT. For a generic M-way partition $\mathbf{B}$, ELT is obtained as:

$$\mathit{ELT}_{\mathbf{B}} = \sum_{j=0,\ldots,M-1} (LT(min_j) \cdot S_j)$$

where min_j and S_j are the line with minimum idleness and the size of block j, respectively. $LT()$ represents the lifetime vs. idleness function of Figure 3.

Given this compact cost function and the fact that M is a small number, it is reasonable to think of an exhaustive exploration algorithm in which all possible p-partitions with $p = 2, \ldots, M$ are evaluated and the one yielding maximum ELT configuration is stored. By representing a M partition as a set of $M-1$ address boundaries, we can generate all possible partitions by enumerating all possible boundaries using a classical recursive backtracking framework.

As the results will show, ELT-driven partitioning alone already yields significant benefits in terms of both aging and energy with respect to a fixed-size partition as the one of [17], thanks to a better matching between the partition sizes and the idleness profile. However, the knowledge of the idleness profile can be exploited so as to further improve both aging and energy, at the cost of a small hardware overhead. The basic transformation we implement is to *selectively swap addresses across partitions* in order to achieve a better overall ELT. This can be easily implemented by modifying the cache indexing function for a few, selected addresses.

The choice of a possible swap-based strategy depends on its relation with the ELT-driven partitioning step. There are essentially two options to combine these two phases.

The first, and most intuitive is to run the partitioning first and then improve the results of partitioning with a set of swaps. We call this strategy **partition & swap**. A second option is to first tweak the idleness profile with a set of swaps and then find the best partition on that profile. We call this strategy **cluster & partition**.

In the following we describe two detailed algorithms for each of the two strategies. Both algorithms are parameterized by a parameter k, which denotes the number of swapped addresses.

1) Partition & Swap Strategy: Since both size and minimum idleness concur to determine ELT, the basic principle behind this strategy is to repeatedly swap the address with the minimum idleness in the largest block with some address (with a larger idleness) of a smaller block that dies earlier.

The operation of the algorithm (called k**-swap**) can be described as follows (see pseudocode): First we get the partition $\mathbf{B} = B_0, \ldots, B_{M-1}$ with sizes $S_0, \ldots, S_{M-1}$. We then repeat k times the swap between the address with maximum idleness in the earliest failing block (i) and the one with the minimum idleness in the block j in which such a swap would maximize the benefit. The latter is defined as the product between size of the block and difference between the second and first minimum ($S_i \cdot (m_{2j} - m_{1j})$. The second factor represents how much the lifetime of this block would be extended.

```
1:  $k$-Swap ($\mathbf{I}$)
2:  $\mathbf{B}$ = ELT-DrivenPartitioning ($\mathbf{I}$)
3:  for $l = 1 \ldots k$ do
4:      $i \Leftarrow$ index of address with $l$-th maximum idleness in
        the earliest failing block.
5:      $j$ = index of block with maximum value of $S_j \cdot (m_{2j} -
        m_{1j})$.
6:      $m_j \Leftarrow$ index of $m_{1j}$
7:      if ($\mathbf{I}[i] > \mathbf{I}[m_j]$) then
8:          SWAP($\mathbf{I}[i], \mathbf{I}[m_j]$)
9:      end if
10: end for
11: return $\mathbf{B}$
```

Clearly, the swap is done only if beneficial (i.e., if we are bringing into Block j an address with idleness higher than the previous minimum m_{1j}).

2) Cluster & Partition Strategy: The rationale behind this strategy is driven by the observation that the ELT-driven partitioning would provide ideal results if the idleness profile $\mathbf{I}$ would be sorted in non-decreasing order.

Since sorting the entire profile would require an excessive number of swaps, the algorithm we implement under this strategy (called k**-min clustering**) identifies the k minima in the idleness profile and swaps them with the addresses at the beginning or at the end of the profile (first or last k addresses), as shown in the pseudocode below.

Then, the ELT-driven partitioning is applied on the modified idleness profile.

```
1: $k$-MinClustering ($\mathbf{I}$)
2: $(j_1, \ldots, j_k) \Leftarrow$ indices of the first $k$ minima
3: $i = 0$
4: for $l = 1 \ldots k$ do
5:     SWAP($\mathbf{I}[i], \mathbf{I}[j_l]$);
6:     $i++$
7: end for
8: $\mathbf{B}$ = ELT-DrivenPartitioning ($\mathbf{I}$)
9: return $\mathbf{B}$
```

Notice that also in this case k is an upper bound on the number of swaps. Some of the minima might already be "in place".

V. Experimental Results

A. Experimental Setup

The proposed architecture has been implemented and tested on a set of traces, extracted from the simulation of the MediaBench suite [19] with an in-house cache simulator. The simulator also includes energy and aging models, derived from an industrial 45nm design kit provided by STMicroelectronics, to provide execution metrics as the miss rate, the energy consumption, and the aging of the simulated cache. As described in Section IV, we define lifetime as the time after which the SNM has decreased by more than 20%. Results refer to the worst case for aging in which it is assumed that a fixed value is stored in each cell.

B. Aging Results

Figures 4 and 5 show a comparison of the proposed algorithms against previous works for a 8kB and 16 kB cache, respectively. They reports average lifetime improvement over a monolithic, power-managed cache and refer to the case of $M=2$ blocks.

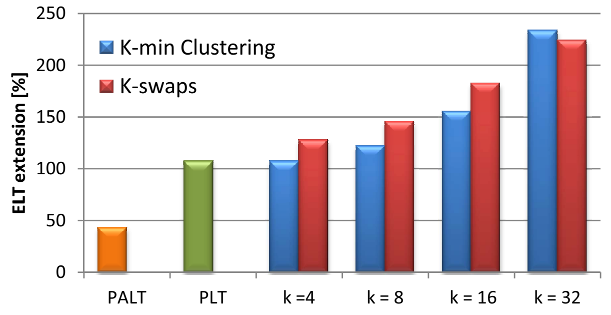

Fig. 4. Lifetime Improvement of 8 KB Cache.

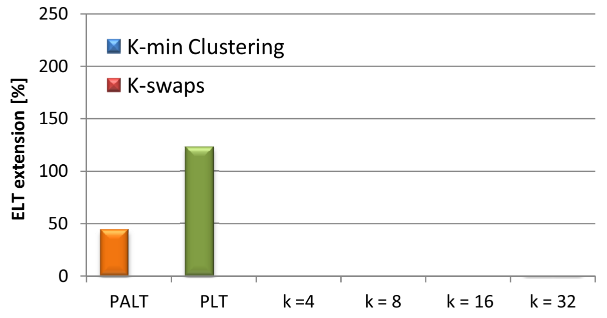

Fig. 5. Lifetime Improvement of 16 KB Cache.

PALT refers to uniform partitioning with re-indexing [17], which yields slightly less than 50% lifetime improvement. PLT denotes the ELT-driven partitioning alone, which provides a lifetime extension of about 120% on average. The adoption of swap-based algorithms yields even better results. Both proposed algorithms perform similarly, although they scale differently with respect to the number of swaps. For smaller number of swaps the k-min clustering does not show considerable improvement in lifetime but then it grows rapidly as k gets larger. Conversely, for the k-swap lifetime increases almost linearly. k-min clustering appears then to be more advantageous for bigger caches with possibility of performing higher number of swaps and k-min clustering will be better suited for situation where only fewer swaps are possible.

In order to analyze the variation of the partitioning efficiency over the benchmarks, Table I reports detailed data.

We can notice that there exists a significant variation across the traces; but for one pathological case (mad) in which only a marginal extension is possible, the benefit is always

TABLE I
DETAILED ELT IMPROVEMENTS [%] (k-SWAP ALGORITHM AND 16 KB CACHE).

	PALT	PLT	k			
			4	8	16	32
adpcm.dec	80.2	248.0	273.5	329.1	498.5	498.5
adpcm.enc	75.3	379.8	425.1	524.3	782.9	782.9
cjpeg	42.4	84.6	103.6	106.4	112.8	116.2
CRC32	8.9	25.8	51.9	63.7	145.7	551.9
dijkstra	89.6	255.6	283.9	288.1	296.3	301.8
djpeg	13.0	28.9	31.3	37.0	81.9	91.7
fft_1	24.6	52.3	62.8	66.3	66.9	69.0
fft_2	17.3	35.8	42.0	47.8	50.1	52.7
gsmd	31.0	94.4	187.2	190.6	196.3	284.4
gsme	76.5	197.0	242.6	245.1	245.2	247.7
ispell	23.4	50.8	59.3	61.3	67.0	72.1
lame	16.6	18.6	19.2	20.6	21.4	22.0
mad	0.6	2.9	7.0	9.5	12.9	14.3
rijndael_i	31.0	69.8	97.4	118.3	142.3	145.9
rijndael_o	46.3	94.7	108.3	145.4	169.6	179.4
say	31.3	79.2	135.5	149.0	161.7	184.2
search	63.8	131.0	141.4	150.2	172.6	189.0
sha	52.3	80.4	95.3	95.6	96.8	103.9
tiff2bw	98.2	400.8	401.4	401.5	401.7	403.8
Average	**43.3**	**122.7**	**145.7**	**160.5**	**195.9**	**226.9**

sizeable. The lifetime benefit is strongly impacted by idleness distribution: applications with higher temporal and spatial locality will provide better results. However, in all cases, results consistently improve those of [17].

C. Energy Results

Table II shows the energy savings for a 16 KB cache, with 16 bytes for line, and for $M = 4$ blocks. We report data for the PLT algorithm and for the k-min algorithm with several values of k.

TABLE II
ENERGY SAVINGS [%] FOR A 4-PARTITIONS WHEN VARYING LINE SWAPS (CACHE SIZE IS 16 KB, LINE SIZE IS 16 BYTES).

	PLT	k			
		4	8	16	32
adpcm.dec	52.4	61.6	57.6	60.7	57.0
adpcm.enc	65.0	55.4	54.6	52.1	62.8
cjpeg	49.9	44.6	48.3	51.9	54.1
CRC32	55.7	54.2	54.4	56.1	65.4
dijkstra	54.0	60.8	60.8	51.8	62.0
djpeg	53.4	51.5	52.1	28.0	38.2
fft_1	44.3	47.8	47.9	48.2	50.1
fft_2	44.2	46.0	46.1	46.5	48.0
gsmd	33.2	40.9	41.6	45.5	52.3
gsme	43.9	40.9	41.1	43.1	43.1
ispell	43.3	44.2	29.0	33.4	38.1
lame	46.7	49.6	49.7	23.3	25.8
mad	52.0	51.3	45.6	46.5	48.2
rijndael_i	54.3	44.9	45.7	47.4	51.9
rijndael_o	45.8	46.1	40.9	46.2	30.8
say	46.6	37.9	41.2	43.4	45.3
search	34.2	42.3	41.8	26.6	46.0
sha	58.0	58.0	56.8	56.3	56.1
tiff2bw	50.1	43.3	42.8	56.6	52.3
Average	**48.8**	**48.5**	**47.3**	**45.5**	**48.8**

The dependency of the energy benefits on the line exchanges is quite irregular. Still, the custom partitioning achieves good results in terms of energy: for each benchmark there is at least one value of k that provides more than 40% of saving. Notice that our approach achieves relevant lifetime extensions with no impact on the energy saving potential (for a 16 KB cache, this approach saves about 48% of the total energy on average, while the uniform partitioning scheme [17] yields about 44%). This slight advantage over [17] sums up to the reduced cost of the hardware overhead required for the decoding circuitry: The static rewiring required to implement the swaps requires far less resources than those needed by a dynamic reindexing scheme, which needs to change the line mapping over time.

TABLE III
AVERAGE ENERGY SAVINGS [%] FOR A 2 AND 4 BLOCKS PARTITIONED CACHE WHEN VARYING CACHE SIZE.

Size (KB)	*2 blocks* PLT	*2 blocks* k = 4	8	16	32	*4 blocks* PLT	*4 blocks* k = 4	8	16	32
4	25.2	18.7	24.0	18.4	18.8	37.3	40.1	45.6	36.2	36.1
8	30.6	19.5	24.9	21.3	23.3	42.7	44.8	50.7	42.9	44.8
16	31.4	23.2	26.4	25.4	28.4	48.8	48.5	47.3	45.5	48.8

Table III shows energy savings, averaged over all the benchmarks, for $M = 2$ and $M = 4$, and for different cache sizes. Clearly, using more blocks or larger caches provides higher saving potential, mainly because the idleness is spread over a larger number of objects (cache partitions or cache lines). Hence the dynamic power manager has more chances to shut down a memory block. Moreover, as M increases, the average dynamic energy cost required for each access decreases

D. Impact on Performance

Figure 6 shows the evolution of the miss rate before and after the "death" of the first block for a 16 KB cache. We report sample curves for a monolithic cache, and for a cache partitioned into 4 blocks using first the PLT and another ones using with the k-min clustering algorithm.

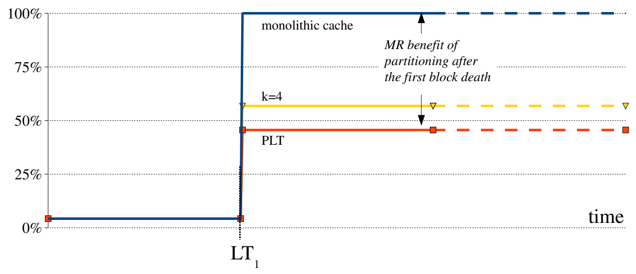

Fig. 6. Miss Rate Before and After the Death of the First Cache Block.

As expected, the miss rate is strongly impacted by the death of the first block. Since we are forcing in that partition the lines used the most, it is very likely that such a block has a relevant impact on cache performance. However, if we compare such a behavior against the one of a monolithic cache (which, from the death of the first block suffers from a 100% miss rate), we can notice that the cache is still working, allowing to exploit the locality principle for more than 50% of the cases.

VI. CONCLUSIONS

We have proposed an application-specific partitioned cache architecture that provides, besides the traditional energy benefits, also significant lifetime extension, thanks to the use of an aging-driven algorithm to drive the calculation of the partition. In addition to the aging-driven partitioning, we introduced an enhancement of the basic algorithm that selectively swap addresses belonging to different blocks to improve the energy and aging quality of the solution.
The two algorithms have similar magnitudes in the aging and energy improvements, but different trends with respect to the number of performed swaps, which make them suitable for different cache sizes. We can achieve aging extensions ranging from 1.2x to 2.3x, depending on the number of swaps and the cache size, while keeping energy smaller by about 50%.

REFERENCES

[1] M.A.Alam, "Reliability- and process-variation aware design of integrated circuits," *Microelectronics Reliability*, Vol. 48, No. 8, August 2008, pp. 1114-1122.
[2] R. Vattikonda, et.al. "Modeling and minimization of PMOS NBTI effect for robust nanometer design," *DAC-44*, pp. 1047-1052, 2006.
[3] S. V. Kumar, et al., "NBTI-Aware Synthesis of Digital Circuits," *DAC-45*, pp. 370–375, June 2007.
[4] Y. Wang et al., "Gate replacement techniques for simultaneous leakage and aging optimization," *DATE'09: Design Automation and Test in Europe*, pp. 328–333, March 2009.
[5] Y. Wang et al., "On the efficacy of input Vector Control to mitigate NBTI effects and leakage power," *ISQED'09: International Symposium on Quality of Electronic Design*, pp. 19–26, March 2009.
[6] K.-C. Wu, D. Marculescu, "Joint logic restructuring and pin reordering against NBTI-induced performance degradation," *DATE'09: Design, Automation and Test in Europe*, March 2009, pp. 75–80.
[7] L. Zhang, R. P. Dick, "Scheduled Voltage Scaling for Increasing Lifetime in the Presence of NBTI," *ASPDAC'09*, pp. 492–497, Jan. 2009.
[8] A. Calimera, E. Macii, M. Poncino, "NBTI-Aware Power Gating for Concurrent Leakage and Aging Optimization", *ISLPED '09: International Symposium on Low power Electronics and Design*, pp. 127-132, August 2009.
[9] S.V. Kumar, K.H. Kim, S.S Sapatnekar, "Impact of NBTI on SRAM read stability and design for reliability," *ISQED'06*, March 2006, pp. 213–218.
[10] Y. Kunitake, T. Sato, H. Yasuura, "A case study of Short Term Cell-Flipping technique for mitigating NBTI degradation on cache," *ISQED'10: International Symposium on Quality Electronic Design*, pp. 660–666, March 2010.
[11] J. Abella, X. Vera, O. Unsal and A. González, "NBTI-Resilient Memory Cells with NAND Gates for Highly-Ported Structures", *Workshop on Dependable and Secure Nanocomputing*, June 2007.
[12] T. Siddiqua, S. Gurumurthi, "Recovery Boosting: A Technique to Enhance NBTI Recovery in SRAM Arrays," *ISVLSI'10: IEEE Annual Symposium on VLSI*, July 2010.
[13] M. Powell, et al. "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," *ISLPED'00: International Symposium on Low power Electronics and Design*, July 2000, pp. 90–95.
[14] K. Flautner, N. Kim, S. Martin, D. Blaauw, T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," *ISCA'02: International Symposium on Computer Architecture*, May 2002, pp. 148–157.
[15] A. Ricketts, J. Singh., K. Ramakrishnan, N. Vijaykrishnan, D. K. Pradhan. "Investigating the Impact of NBTI on Different Power Saving Cache Strategies," *DATE'10: Design, Automation and Test in Europe*, pp. 592–597, March 2010.
[16] A. Calimera, M. Loghi, E. Macii, M. Poncino, " Dynamic indexing: Concurrent leakage and aging optimization for caches", *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pp.343-348, 18-20 Aug. 2010
[17] A. Calimera, M. Loghi, E. Macii, M. Poncino, " Partitioned cache architectures for reduced NBTI-induced aging", *DATE 2011: Design Automation and Test in Europe*, pp. 938-943, March 2011.
[18] Z. Qi, et al. "SRAM-based NBTI/PBTI sensor system design," *DAC-47: 47th Design Automation Conference*, June 2010, pp. 48.1–48.4.
[19] M. R. Guthaus et al., "MiBench: A free, commercially representative embedded benchmark suite", *IEEE 4th Annual Workshop on Workload Characterization*, pp. 3–14, Dec. 2001.