Buffering of Frequent Accesses for Reduced Cache Aging

Andrea Calimera[†], Mirko Loghi[‡], Enrico Macii[†], Massimo Poncino[†]
[†]Politecnico di Torino, 10129, Torino, ITALY
[‡]Universitå di Udine, 33100, Udine, ITALY

ABSTRACT

Previous works have shown that typical power management knobs such as voltage scaling or power gating can also be exploited to reduce aging phenomena caused by Negative Bias Temperature Instability (NBTI).
We propose a scheme for power-managed caches that allows to significantly improving the aging of the cache thanks to the use of a small buffer that stores a copy of the lines that are most critical for aging, that is, the ones with the least opportunity of being power-managed; by using the buffer instead of the cache when accessing these critical lines, the original cache is preserved and its lifetime is significantly prolonged. As a side effect, this scheme improves total power since the less energy-hungry buffer is accessed most of the time. Experimental analysis shows this scheme allows to achieve significant ($> 3x$ on average) lifetime extensions for the cache, with a concurrent energy saving between 18 and 24%, depending on cache size.
Categories and Subject Descriptors: B.3.2 [MEMORY STRUCTURES] : Design Styles.
General Terms: Design, Experimentation, Performance.
Keywords: Memory Hierarchy, Leakage Reduction, Aging.

1. INTRODUCTION

The most relevant source of device aging in sub-65nm technologies is represented by Negative Bias Temperature Instability (NBTI) [1] which affects pMOS devices under negative bias (i.e., $V_{gs} < 0$, i.e., when a "0" is applied on the gate input of a pMOS, called the *stress* condition); under this condition, a temporal drift of the threshold voltage occurs, which translates into a increase of the propagation delay over time. Such a drift is partially mitigated by the application of a logic "1" (the *recovery* condition), which decreases the threshold voltage and thus partially recovers the delay [1].
These aging effects are particularly hard to tackle in SRAM memories, for two reasons. First, unlike logic circuits, the aging of the two inverters of the bitcell affects the *stability* of the cell (that is, the capability of a cell of safely storing a value), rather than its *delay*. Second, due to the symmetric structure of a SRAM cell, a SRAM cell ages in fact whatever the value it stores; therefore, there is no true opportunity of recovery.
Some works have shown that typical implementations of power management (namely, by means of *voltage scaling* or *power gating*) can be exploited for alleviating the aging. *Voltage scaling* is effective because a smaller V_{dd} corresponds into a smaller V_{gs}, and thus in a smaller magnitude of negative bias [7]. *Power gating*, when implemented through a footer transistor, becomes a powerful knob: the disconnection of a logic block from the ground network pulls all internal nodes inside the block to a logic "1", thus completely nullifying the aging [8].
Unlike conventional aging-improving strategies used for logic circuits, like value management [2][3] or guard-banding [4]–[6], power management is naturally applicable to memory cells, and by aggregation, to diversely sized portions of a memory (e.g., a row, a column, or a bi-dimensional region); we call this portion the *unit of power management (UPM)*. The consequence of this observation is therefore that the *idleness* resulting in typical cache access patterns [9, 10] is synonymous of both power **and** aging reduction. This property holds however only for the individual UPMs; when evaluating the aggregate benefit over the whole memory, the different natures of power (a cumulative cost function) and aging (a worst-case one) becomes evident. Each power-managed UPM will in fact contribute to the total power saving with its one (small or large) contribution; conversely, from the aging standpoint, the first failing UPM (the one with the least power management opportunities) will cause the entire memory to become unusable. In order to exploit idleness for reducing both power and aging for a memory block, it is therefore essential to re-shape the memory access distribution in such a way that the average (over the UPMs) idleness and the worst-case idleness (the UPM with the least idleness) are as similar as possible.
In this work we focus on caches in which the UPM is an individual cache line, and propose the use of a small (typically a few tens of entries) buffer in parallel to the main cache, which stores a copy of the cache lines having worst-case access patterns. Whenever one of these lines is accessed, the access is redirected to the buffer so that the corresponding line in the main cache is always kept unused (and thus in a standby state) for the whole duration of the workload.
Unlike previous approaches [16], in which perfect uniformity of the access patterns is achieved through a time-varying addressing scheme (*dynamic indexing*), in our approach we

selectively smooth out the worst-case patterns by duplicating selected addresses. In this way we achieve a comparable aging benefit but we also save energy (static and dynamic) with respect to [16]. The lines with the worst-case idleness are in fact also the most accessed ones, so the buffer can be seen as a level-0 cache, from the performance standpoint.
The scheme is clearly application-specific, since it requires the profiling of the memory access patterns to detect lines with the least idleness. An important dimension of the proposed methodology is also how many lines have to be copied in the buffer; since there is an intuitive tradeoff between energy saving and the aging benefits, we define energy and aging models as a function of the number of the critical lines to be copied that allows to find the optimal number of lines copied into the buffer for a given application.
Results show that our buffering scheme allows to significantly improve the lifetime of power-managed caches: by copying on average about 6% of the cache lines (for a 16KB cache), our scheme improves aging by 3.4X with respect to a standard cache, with a 24% improvement in total energy. We also show how it is possible to use a fixed size (namely, 16 lines) for the buffer, thus restricting the application dependence to the calculation of the critical lines, while allowing the use of a fixed architecture.

2. BACKGROUND AND RELATED WORK

2.1 Background

For an in-depth analysis of NBTI effects and models we refer the reader to classical tutorial papers on NBTI (e.g., [1]). We summarize here the basic issues involved in NBTI-induced aging in SRAM cells and arrays.
Due to its symmetric structure, a SRAM cell ages whatever the value stored: therefore, there is no preferential value and no equivalent of recovery. Value dependency matters only in terms of what value is predominant: the best-case degradation occurs in fact when the pMOS of both inverters in the cell age for the same amount of time, that is, when the bitcell stores a 0 and a 1 with equal probability [3].
Another important aspect is that the aging of the two bitcell inverters does not truly affect the *delay* of the cell but rather its stability. The conventional metric for assessing the aging of a SRAM cell is the Static Noise Margin (SNM), defined as the minimum DC noise voltage necessary to change the state of an SRAM cell; when the SNM of a cell falls below a threshold that allows safe storage of data it cannot be safely read or written. This threshold strongly depends on the technology and the specific design of the memory cell (e.g., transistor W/L ratios, threshold voltages, etc.).

2.2 Related Work

Aging due to NBTI is usually tackled by following two main approaches: (i) *guard-banding* strategies, where the circuit is over-designed under a delay constraint tighter than the nominal one (e.g., by increasing supply voltage, using larger or low-threshold devices), so that the aging is compensated usually at the price of other metrics ([4]–[6]); (ii) *mitigating* solutions, which act directly on the variables that affect NBTI aging: threshold and/or supply voltage, gate sizes, and signal probabilities ([2]–[4]).
Other solutions do not fit the above classification and combine power (static, in particular) and aging reduction by exploiting the existence of a low-power "standby" state for the circuit under analysis. This low-power state can be exploited either by using a sort of "gated" version of standard library cells thus allowing to minimize the number of logic 0's in the circuit [11], or by using special vectors to be applied during standby [12].
When moving to SRAMs, approaches attempt at maximizing (structurally or functionally) the conditions under which the degradation in a memory cell is minimal, i.e., a 50% probability of storing a value. The approach of [3] proposes hardware and software schemes to periodically invert the entire content of a memory so as to guarantee a perfectly balanced bitcell probability.
The method of [13] proposes a new design of a memory cell consisting of a set of NAND gates arranged in such a way that minimum degradation ratio for all pMOS transistors in the cell can be obtained.
A recent category of approaches relies on the exploitation of the joint benefits of power management implementations for aging mentioned in Section 1. Specifically, the works of [14, 15] have quantitatively assessed this benefit on entire memory blocks for voltage scaling and power-gating, respectively.
The work of [16] introduces the idea of a time-varying addressing scheme for caches (*dynamic indexing*); in this work, the UPM is a single cache line, and their method allows achieving identical lifetime for all cache lines and, by extension, maximal lifetime for the entire cache.

3. FREQUENT ACCESS BUFFERING

3.1 Motivation

As mentioned in Section 1, in order to constructively exploit the existing idleness also for aging, the distribution of idleness should be as uniform as possible. Unfortunately, this is not usually the case: such a distribution is highly non-uniform. Figure 1 shows the distribution of idleness (percentage of time a line can be turned off) for an example cache with 256 line running one of the traces used in our experiments.

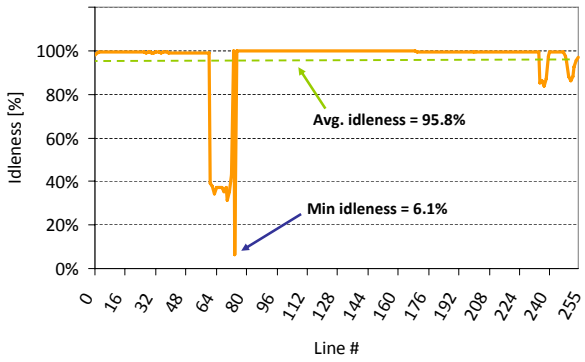

Figure 1: Example of Idleness Distribution.

We can observe that the average idleness is very high (95.8%), which roughly translates into an equivalent leakage reduction. We notice however that there exists a set of adjacent lines with much lower idleness (Lines 60–75); one in particular appears to be turned off only 6.1% of the time. This line will allow exploiting only about 6% of the overall idleness.

Our idea is that of selectively copying such "critical" lines into a small side buffer and access the buffer instead of the cache. Figure 2 shows how aging can be improved by copying to the buffer the most critical lines; in particular, the plot shows how the benefit increases (non linearly) versus the number of buffered lines. For instance, after buffering only 1 line (the one with only 6% idleness), the worst case line has an idleness of about 31%; buffering other 12 lines does not provide significant improvement, but a 13th line will rapidly bring the minimum idleness to 83.4%.

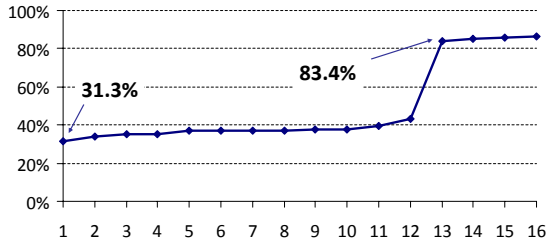

Figure 2: Aging Reduction vs. # of Buffered Lines.

Copying lines into the buffer will incur into an energy overhead that has to be evaluated against the aging benefits; it is likely that there will be an optimal energy-aging tradeoff point. From this example, it is clear that this solution is strongly application specific: each workload will require a different set of lines to be buffered.

3.2 Architectural Details

Figure 3 shows an abstract architecture of the proposed buffering scheme. The index portion of the cache address (n bits) is decoded by the selector block *Sel*, which detects whether the index is one of the r line addresses stored in the buffer. If so, it deactivates the main cache and it uses the buffer. The matching index is translated into a buffer address on $\lceil \log_2 r \rceil$ bits.

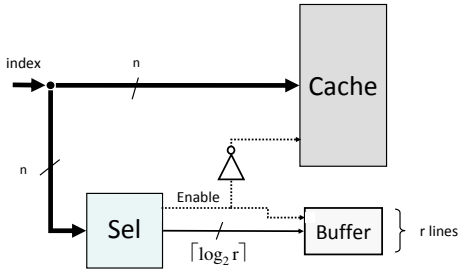

Figure 3: Overall Architecture.

The *Sel* block conceptually works as a lookup table; as such, it could be implemented as a small, fully associative cache. This option would however be unnecessarily complicated; for our purposes, the selector is always used for reading (but for the offline upload of the target addresses), so the control circuitry to allow fully associative write accesses would be wasted; furthermore, we do not need the *address value*, but rather we need to translate, in case of match, the matched value into a buffer address on $\lceil \log_2 r \rceil$ bits.
For these reasons, we adopted a customized implementation (Figure 4) where the r line addresses are stored in registers, and the incoming addressed is compared in parallel to the n line addresses.

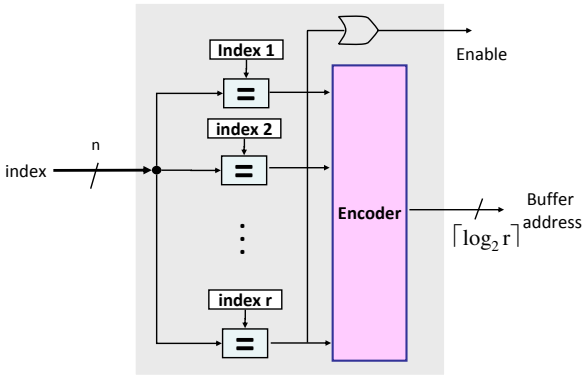

Figure 4: Internal Structure of the Buffer Decoder.

Only one of the comparators will match, and the r outputs of the comparators represent a 1-hot encoded signal. A priority encoder (the *Encoder* block) translates this r-bit value into a $\lceil \log_2 r \rceil$-bit address for the buffer. The buffer itself is a conventional direct-mapped cache, i.e., it also contains a tag array and the required control logic to manage the lookup results (hit/miss).
Notice that implementing power management inside the buffer is not very useful. Since the critical lines are also the most accessed ones, the buffer will be frequently used in place of the main cache so there will be few opportunities of power management.
Finally, OR-ing of the r comparator outputs will then generate the activation/de-activation signal.

3.3 Functional Operations

After system profiling has been run, the r most critical line addresses $(0, \ldots, 2^n - 1)$ are stored in the registers of the encoders. The buffer is initially empty, and it gets filled as we start accessing the critical lines.
Hits and misses are managed as usual in the both the main cache and the buffer. In the main cache only non critical lines will be accessed. In the buffer, whenever a critical line index is detected, the proper address is generated and conventional cache lookup occurs. If tag matches, a hit occurs; otherwise, the miss procedure is started. A line will be fetched in the next level of hierarchy and copied into that location.
Technically, the approach is very much like two caches in parallel yet mutually exclusive are used, one being much smaller than the other.

3.4 Models and Optimization

A careful analysis of the tradeoff existing between the number of lines r copied to the buffer and the energy and area overhead is needed in order to come up with the optimal value of r for a given workload. In the following paragraphs we review the impact of r on the various metrics.

Aging.
The impact on aging clearly depends on the shape of the distribution and it is in general non linear, as shown in Figure 2. Nevertheless, the benefit is a monotonically increasing function of r: adding a single line to the buffer will reduce aging by a quantity equivalent to the incremental benefit of the moved line (i.e., the difference of idleness). Let us de-

note this lifetime function with $LT(r)$; the evaluation of this function is clearly strongly dependent on the workload.
For a given workload, $LT(r)$ can be easily obtained by observing the distribution of idleness (i.e., Figure 1). Let $\mathbf{I} = \{I_1, \ldots, I_n\}$ the list of idleness values of each of the n lines, sorted in non-decreasing order. If the buffer has r locations, it will store the lines with the r smallest values of idleness, i.e., $I_1, \ldots, I_r$. After these are copied in the buffer, their location in the main cache will not be accessed anymore, and the main cache will become unreliable approximately when the line with idleness I_{r+1} fails. The benefit in lifetime $\Delta LT(r)$ is therefore given by $\Delta LT(r) = \sum_{j=1,\ldots r} I_j$.

Power.
The selection circuitry of Figure 4 will consume extra dynamic power with respect to the original architecture. This power is approximately linear in r: the addition of a line requires one extra register, one comparator, and one extra line in the buffer, plus a few gates in the encoder. Let $P_{dyn,sel}(r)$ this power value (which is a systematic penalty).
On the other hand, however, we are accessing the buffer (a smaller and thus less power hungry memory) most frequently. Therefore, the actual overhead in dynamic power depends also on how often we use the buffer instead of the main cache, and this value (a sort of "hit"rate) clearly depends on r. If we call $\pi(r)$ the probability of accessing the buffer, the difference in power from using the buffer is $\pi(r) \cdot (P_{buf}(r) - P_{cache})$. Since $P_{cache} > P_{buf}(r)$ for $r < 2^n$, this overhead is actually a benefit.
Leakage power, conversely, is roughly identical to the baseline case. The leakage overhead caused by the extra logic $P_{static,enc}(r)$ is relatively small since all the elements are active most of the time: the selection circuit is accessed in any cycle and the buffer is used most of the time in place of the main cache. The leakage in the memory elements is unchanged: lines that were leaking in the main cache will leak in the buffer by exactly the same amount.
The overall variation in power is therefore $\Delta P_{tot}(r) = (P_{dyn,sel}(r) + P_{static,sel}(r)) + \pi(r) \cdot (P_{buf}(r) - P_{cache})$.

Performance.
For performance we have to distinguish between worst case access and average access time. The former is determined by the case in which we access the main cache; in this case, the propagation delay of the selection logic $t_{p,sel}(r)$ is the propagation delay adds up to the cache access time. As we will show later, this value is a small fraction of the cache cycle time. Notice that the dependence on r of $t_{p,sel}$ is quite weak due to its parallel implementation.
The average access time, conversely, is improved, using the same arguments discussed for dynamic power. The average access time can then be written as $T_{acc,avg}(r) = (t_{p,sel}(r) + \pi(r) \cdot t_{acc,buf}(r) + (1 - \pi(r)) \cdot t_{cache}))$.
The improvement in average access time is then $T_{acc,avg}(r) - t_{cache} = t_{p,sel}(r) + \pi(r) \cdot (t_{acc,buf}(r) - t_{cache})$.
In our tradeoff analysis we aim at finding the best power/lifetime point, while performance is regarded as a side metric. More precisely, we explore the *power- lifetime product* (PLP), defined by the product of the respective benefits (power saving and lifetime improvement), that is:

$$PLP(r) = \frac{\Delta LT(r)}{LT} \cdot \frac{\Delta P_{tot}(r)}{P}$$

where LT and P are lifetime and power for the baseline configuration, respectively.

Figure 5 qualitatively shows a typical tradeoff scenario.

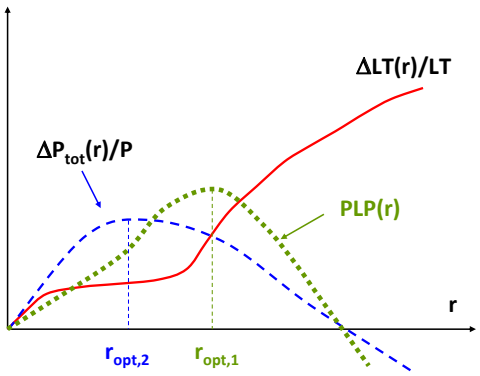

Figure 5: Power/Lifetime Tradeoff.

The lifetime improvement $\frac{\Delta LT(r)}{LT}$ grows monotonically with r. The power saving $\frac{\Delta P_{tot}(r)}{P}$, conversely, is a benefit for small values of r. As r increases, the fixed cost due to the selector $((P_{dyn,sel}(r) + P_{static,sel}(r)))$ gets larger and the dynamic power reduction $(P_{buf}(r) - P_{cache})$ due to the use of smaller buffer tends to tail off; as a result, the power saving becomes a penalty. There is therefore an optimal value $r_{opt,2}$ of r that minimizes the total power.
Since the PLP is the multiplication of the power saving by a monotonically increasing function, it will also have a maximum in correspondence of some value $r_{opt,1}$. Since the range of values for r is limited, the exploration is done in a straightforward way by evaluating $PLP(r)$ for all values of $r = 0, \ldots, 2^n - 1$.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

The proposed architecture has been implemented and tested on a set of traces extracted from the simulation of the MediaBench suite [17]. We used a cache simulator that mimics the behavior of a cache, also taking into account the energy consumption for each access. Such a consumption is computed by leveraging power/energy models derived from an industrial 45nm design kit provided by STMicroelectronics. Concerning aging characterization of memories, we implemented a dedicated SPICE-based characterization framework which predicts, under user-defined PVT operating conditions, the aging profile of a 6T-SRAM cell. Characterization is carried out for various physical parameters of the cell (netlist, size of the transistors, process parameters) as well as for functional ones (the probability of storing a '0' logic, and the idleness – % of time in standby state – of the cell). The characterization process first calculates, based on HSPICE built-in aging models, fitted to the parameters of our technology library, the threshold voltage degradation ΔV_{th} of each transistor in the cell. This variation is annotated into the SRAM cell netlist as DC-controlled voltage source on the gate terminal of each pMOS transistor as done in [18]. In a second phase, the annotated cell is simulated to extract the SNM (specifically, the *read* SNM, when both access nMOS transistors are on); The collected SNM data are stored in a lookup table, which is used by the cache simulator to estimate the aging of the cache lines.

In the experiments, lifetime of a memory cell is defined as *the time after which the SNM has decreased by 20%.*

4.2 Overhead Assessment

We synthesized several configurations of the buffer decoder in order to estimate the power consumption and the propagation delay for different values of r.
Figure 6 shows the overhead in terms of total power consumption and additional delay time for a 8kB cache, when varying the size of the buffer (r).

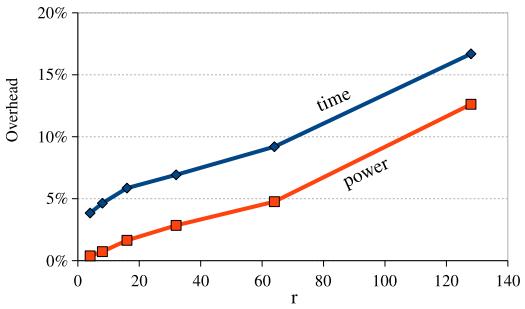

Figure 6: Overhead of the buffer decoder.

The power overhead is reasonable, and it can be easily compensated by the energy benefits provided by the buffer. The timing overhead, instead, becomes sizable for high values of r and, thus, can pose an upper limit to the size of the buffer (which depends on the slack time originally available in the memory subsystem).

4.3 Power-Lifetime Product Optimization

Table 1 shows the power savings and the lifetime extension for each benchmark and for three different cache sizes. In this first set of results, we chose for each trace the value that maximizes the power-lifetime product (r_{opt_1} in Figure 5).

	4kB			8kB			16kB		
	P_{sav} [%]	LT $Ext.$	r	P_{sav} [%]	LT $Ext.$	r	P_{sav} [%]	LT $Ext.$	r
adpcm.dec	38.4	2.1x	14	39.9	2.1x	20	43.7	2.2x	20
adpcm.enc	39.3	2.5x	18	40.9	2.7x	22	43.8	3.0x	24
cjpeg	15.5	3.5x	55	18.4	3.7x	80	21.0	3.8x	85
CRC32	44.5	2.5x	26	47.9	2.7x	26	49.4	2.9x	26
dijkstra	23.8	3.0x	14	28.1	2.9x	16	30.6	2.9x	29
djpeg	11.9	3.5x	46	13.6	3.7x	81	16.3	3.8x	91
fft_1	11.3	3.5x	53	14.9	3.7x	100	19.9	3.8x	115
fft_2	9.1	3.5x	58	12.1	3.7x	77	14.9	3.8x	139
gsmd	29.0	2.7x	39	34.1	2.7x	47	37.0	2.9x	55
gsme	23.0	3.2x	51	22.8	3.2x	52	23.5	3.2x	58
ispell	9.5	3.6x	60	12.2	3.7x	81	15.0	3.8x	82
lame	6.7	3.5x	56	8.5	3.7x	89	7.7	3.8x	109
mad	4.5	3.6x	50	6.2	3.7x	99	7.3	3.8x	102
rijndael_i	4.8	3.5x	22	6.1	3.7x	25	7.7	3.8x	28
rijndael_o	5.2	3.4x	22	6.2	3.6x	20	7.6	3.8x	27
say	12.6	3.5x	63	15.8	3.7x	85	20.9	3.8x	121
search	14.0	3.6x	46	16.4	3.6x	49	17.3	3.6x	50
sha	29.5	3.7x	63	37.4	3.8x	63	40.8	3.9x	69
tiff2bw	13.3	3.6x	11	15.7	3.4x	11	23.7	3.1x	10
Average	**18.2**	**3.3x**	–	**20.9**	**3.4x**	–	**23.6**	**3.5x**	–

Table 1: Total Power Savings and Lifetime Extension: Power-Lifetime Product Optimization (Line Size is 16 Bytes).

Since applications exhibit very diverse access profiles, the number of lines that must be placed into the buffer is quite irregular and spans from 10 (for `tiff2bw`, where the buffer replaces less than 1% of the main cache) to 139 (for `fft_2`, where the buffer size is about 13% of the cache size).
Nevertheless, we can observe a quite satisfactory result in terms of aging mitigation for every benchmark; we have at least a 2x improvement in the cache lifetime of the cache, and about 3.3x–3.5x average improvement on average, depending on cache size. Notice that 2x lifetime extension would be the result obtainable by a naive scheme in which the cache is duplicated, the first one being used until it becomes unreliable and moving then to use the second one. Results show that a "smart" duplication of lines allows to get much higher average benefits at a much lower overhead. It is also worth observing that the benefits scale well: figures improve as cache size increases.
While aging benefits have a relatively small variance, it is also interesting to observe how power savings have a significant variation instead. This is due to the very heterogeneous application characteristics of the traces. Applications with a small subset of lines that are heavily stressed (e.g., `CRC32`) benefit of a considerable power reduction by leveraging a small buffer to resolve a large fraction of accesses. Conversely, applications with a more uniform idleness profile (such as `mad` or `lame`) do not achieve a significant power benefit, since the overhead becomes significant.
In any case, even in the most unfavorable cases the proposed strategy allows to extend the lifetime span without incurring in any power penalty; rather an average 18 to 23% (depending on cache size) power reduction is obtained.

4.4 Power minimization

Since the buffer leads to a more power efficient structure in many cases, we also investigated an exploration strategy that considers only power savings. This corresponds to the point labeled as $r_{opt,2}$ in Figure 5. Table 2 show power (and lifetime) results when such a
$r = r_{opt_2}$ is chosen.
Albeit in several cases the choice for r does not change, some applications (e.g., `cjpeg` and `sha`) can obtain a slightly better power saving and a reduced buffer size, at the cost of a small reduction of the lifetime.
In some circumstances, however, looking for the maximum power saving can strongly deteriorate the lifetime of the cache. That is the case of `adpcm.dec` over a 4kB cache, where removing one single line from the buffer (17 vs. 18) causes a reduction of the lifetime extension from 2.5x to 1.6x. Such a situation is the symptom that the line left into the main cache is too frequently addressed and, thus, leads to a fast aging of the memory.

4.5 Relaxing the Application Dependency

The previous tables were referring to results obtained by using the optimal r for each benchmark, this implies that a customized, application-specific architecture would be required. Therefore, the benefits of our scheme would be less appealing for systems that run more than an application. The solution would be that of choosing a fixed value for r that is good on average and can still provide significant benefit (even if not the best possible) for a heterogeneous set of benchmarks. In Table 3 we report results for such a kind of structure, where r is arbitrarily fixed to 16.
Results are somehow surprising. But for a couple of applications, that do not reach the threshold of a 2x lifetime extension, a 16-entry buffer provides a significant improvement

	4kB			8kB			16kB		
	P_{sav} [%]	LT $Ext.$	r	P_{sav} [%]	LT $Ext.$	r	P_{sav} [%]	LT $Ext.$	r
adpcm.dec	38.4	2.1x	14	39.9	2.1x	20	43.7	2.2x	20
adpcm.enc	39.4	1.6x	17	40.9	2.7x	22	43.8	3.0x	24
cjpeg	15.6	3.4x	51	18.8	3.6x	48	21.0	3.8x	85
CRC32	44.7	1.9x	25	48.1	2.1x	25	49.5	2.4x	25
dijkstra	23.8	3.0x	14	28.1	2.9x	16	30.6	2.9x	29
djpeg	12.1	3.4x	33	13.6	3.6x	45	16.3	3.8x	91
fft_1	11.3	3.5x	53	14.9	3.7x	90	19.9	3.8x	111
fft_2	9.1	3.5x	56	12.1	3.7x	77	14.9	3.8x	139
gsmd	29.0	2.7x	39	34.1	2.7x	47	37.0	2.9x	55
gsme	23.0	3.2x	51	22.8	3.2x	52	23.5	3.2x	58
ispell	9.5	3.6x	60	12.3	3.7x	70	15.0	3.8x	82
lame	6.7	3.5x	56	8.5	3.7x	89	7.7	3.8x	108
mad	4.5	3.5x	38	6.2	3.7x	94	7.3	3.8x	95
rijndael_i	4.8	3.5x	22	6.2	3.5x	20	7.7	3.8x	28
rijndael_o	5.2	3.4x	22	6.2	3.6x	20	7.6	3.7x	24
say	12.6	3.5x	60	15.8	3.7x	85	20.9	3.8x	121
search	14.1	3.5x	41	16.4	3.6x	49	17.3	3.6x	50
sha	30.5	3.6x	57	37.8	3.6x	57	41.2	3.8x	60
tiff2bw	13.4	3.2x	10	15.8	3.2x	10	23.7	3.1x	10
Average	**18.3**	**3.1x**	–	**21.0**	**3.3x**	–	**23.6**	**3.4x**	–

Table 2: Total Power Savings and Lifetime Extension: Power-Only Optimization (Line Size is 16 Bytes).

	4kB		8kB		16kB	
	P_{sav}[%]	LT_{ext}	P_{sav}[%]	LT_{ext}	P_{sav}[%]	LT_{ext}
adpcm.dec	38.4	2.1x	39.7	2.1x	43.4	2.2x
adpcm.enc	38.9	1.6x	40.2	1.9x	42.8	2.1x
cjpeg	11.5	2.9x	13.4	3.0x	14.8	3.1x
CRC32	37.6	1.8x	39.5	2.0x	40.3	2.2x
dijkstra	23.8	3.0x	28.1	2.9x	30.3	2.9x
djpeg	11.2	2.7x	11.9	2.9x	12.8	3.0x
fft_1	7.0	3.1x	7.2	3.3x	8.5	3.4x
fft_2	5.5	3.2x	5.9	3.3x	6.3	3.4x
gsmd	16.8	2.5x	18.1	2.7x	19.2	2.9x
gsme	14.9	3.0x	14.4	3.2x	14.0	3.2x
ispell	6.9	3.3x	7.5	3.4x	7.7	3.5x
lame	3.9	3.3x	4.1	3.5x	3.7	3.6x
mad	3.2	3.4x	2.9	3.5x	3.4	3.6x
rijndael_i	4.1	3.4x	4.3	3.4x	5.9	3.5x
rijndael_o	4.4	3.4x	5.7	3.5x	6.7	3.5x
say	7.5	2.8x	8.3	3.0x	9.7	3.1x
search	11.9	3.3x	13.0	3.4x	14.0	3.5x
sha	15.2	3.0x	18.6	3.0x	18.5	3.0x
tiff2bw	12.9	3.6x	15.3	3.4x	23.3	3.1x
Average	**14.5**	**2.9x**	**15.7**	**3.0x**	**17.1**	**3.1x**

Table 3: Power Savings and Lifetime Extension for a Fixed 16-Entry Buffer (Line Size is 16 Bytes).

both in terms of power saving and of aging mitigation. Average lifetime extension is still greater than 3x for all cache sizes, and power savings are slightly increased. This behavior is a sign of the fact that the PLP curves are in most cases pretty flat around the maximum; therefore, even choosing a value of r far off the optimum $r_{opt,1}$ does not degrade the cost significantly.
Such a result enforces the principle that the most of the aging is accountable to a small portion of the cache, in a novel perspective of the principle of locality.

5. CONCLUSIONS

We proposed an application-specific cache architecture that relies on the use of a small buffer alongside the main cache, which contains a copy of the cache lines with worst-case access profile. This arrangement allows to indirectly remove "stress" from the corresponding lines in the main cache by increasing the opportunities of power managing them. In this way, the lifetime of the main cache is prolonged significantly, with the additional side benefit of a non negligible energy saving, due to the fact that lines in the buffer are typically also the most accessed ones.
This architecture provides average cache aging improvements of more than 3x with more than 20% energy saving. Moreover, the idea scales nicely, as larger caches also provide larger benefits.

6. 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] S. V. Kumar, et al., "NBTI-Aware Synthesis of Digital Circuits," *DAC-45*, pp. 370–375, June 2007.

[3] 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.

[4] R. Vattikonda, et.al. "Modeling and minimization of pMOS NBTI effect for robust nanometer design," *DAC-44*, pp. 1047-1052, 2006.

[5] X. Yang, K. Saluja, "Combating NBTI Degradation via Gate Sizing," *ISQED'07: International Symposium on Quality Electronic Design*, pp. 47–52, March 2007.

[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*, pp. 75–80, March 2009.

[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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] A. Calimera, M. Loghi, E. Macii, M. Poncino, "Aging Effects of Leakage Optimizations for Caches," *GLSVLSI'10: IEEE Great Lakes Symposium on VLSI*, pp. 95–98, May 2010.

[16] A. Calimera, M. Loghi, E. Macii, M. Poncino, "Dynamic indexing: concurrent leakage and aging optimization for caches," *ISLPED'10: International Symposium on Low power Electronics and Design*, pp. 343–348, August 2010.

[17] 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.

[18] K.Kang, H. Kufluoglu, K. Roy, M.A. Alam, "Impact of Negative-Bias Temperature Instability in Nanoscale SRAM Array: Modeling and Analysis," *IEEE Transactions on CAD*, Vol. 26, No. 10, pp. 1770-1781, Oct. 2008.