Cache Aging Reduction with Improved Performance using Dynamically Re-sizable Cache

Haroon Mahmood, Massimo Poncino, Enrico Macii
Departimento di Automatica e Informatica, Politecnico di Torino, Torino, ITALY
Email: {haroon.mahmood, massimo.poncino, enrico.macii}@polito.it

***Abstract*—Aging of transistors is a limiting factor for long term reliability of devices in sub-100nm technologies. It's a worst-case metric where the lifetime of a device is determined by the earliest failing component. Impact is more serious on memory arrays, where failure of a single SRAM cell would cause the failure of the whole system. Previous works have shown that partitioning based strategies based on power management techniques can effectively control aging effects and can extend lifetime of the cache significantly. However, such a benefit comes as a trade-off with performance which reduces proportionally as the time elapses. To address this problem and provide a single solution to concurrently improve aging, energy and performance of the cache, we propose an architectural solution based on the dynamically re-sizable cache [5] and cache partitioning approaches. By this strategy, cache is dynamically re-sized and reconfigured whenever a cache block becomes unreliable. Coupling such aging mitigation technique along with dynamically re-sizable cache approach provides on average 30% lifetime improvement with less than 0.4x degradation in performance whereas, in previous solutions, performance degradation sometimes goes upto 10x.**

I. Introduction

Aggressive scaling of CMOS technologies has steadily improved process performance over the years along with reduction in the size of these devices. However, due to power density and increased temperature, various types of non-idealities have emerged in scaled technologies. One of them is device aging which affects the robustness of device operations and deteriorate their performance over time. Among various sources of device aging, Negative Bias Temperature Instability (NBTI) has been identified as the most critical challenge in sub-65 nanometer technologies. NBTI affects PMOS transistors by causing temporal drift of threshold voltage over time with negative gate to source voltage (i.e. a logic "0" on the gate input) which in turn increases the propagation delay of device. In recent years, several works have addressed the issue of NBTI-induced aging by controlling the design variables that regulate the aging process and in particular the logic values [1], [2]. Such solutions are not feasible for SRAM memories due to their symmetric structure: an SRAM cell ages regardless of the value being stored in it. In alternative, partitioning-based strategies coupled with power management techniques proved effective to reduce aging effect in SRAM memories. For instance, the work of [3] proposed a multi-bank architecture that allows different cache sub-blocks to age at different rates thus providing significant improvement in both energy and aging. However the focus of such works remained on aging and energy optimization, while performance of the cache decreases with time due to the fact that first-dying partition is also the one with maximum accesses. So as soon as a partition dies, it causes a sudden increase in the miss rate of the cache and thus reduces its performance significantly.

978-3-9815370-2-4/DATE14/©2014 EDAA

In this work, we have adopted a new approach to tackle the issue of cache performance degradation which is based on dynamically re-sizable cache (DRC) [5] and on the cache partitioning approach (PLT) [3]. The solution proposed in [5] though provides a good solution for energy reduction but for aging it is only partially effective. Aging is a worst-case metric and lifetime of the cache depends on the line with least idleness. And in this approach, it is possible that a specific line is continuously in use and therefore re-sizing does not have any effect on aging. On the other hand, cache partitioning approach [3] provides excellent solution to reduce aging and energy but in this case, performance of the cache reduces significantly as discussed before.

In this work, we specifically target application-specific systems to concurrently improve aging, energy and performance of direct-mapped caches. Our idea is based on the observation that cache idleness profile always exhibits an uneven distribution of idleness. Some lines are heavily accessed and therefore age much more rapidly as compared to a bigger portion of the cache having fewer accesses and thus less prone to aging effect. So in our approach when some portion of the cache is dead, we discard that specific block and re-size the cache to utilize the remaining healthy portion of the cache. Unlike the original DRC, we do not require a continuous analysis of cache workload and repetitively adjust its size; cache works normally until a line is dead which can be detected easily using a sensor proposed in [6] and at that point the cache will be re-sized by discarding the dead cache block. Consequently, the lifetime of the cache consists of two phases. In the first phase, cache works normally with its full potential until a line becomes unreliable which will also mark the end of its original lifetime. Then in second phase, the cache is reconfigured to work as a smaller size cache which is done by remapping the addresses from memory to cache lines. Remapping the addresses will redirect almost all memory accesses back to cache whereas all of them were causing cache misses in the work of [3]. This implies that there will only be a marginal increase in miss rate whereas in previous techniques, miss rate rises exponentially when a partition dies. On average our DRC approach limits the performance degradation to 0.4x as compared to PLT where average performance degradation was 7x.

II. Background and Related Work

In an SRAM cell, threshold voltage drift caused by NBTI does not truly affects the delay but rather affects the stability of the cell. A conventionally accepted metric for the aging of

an SRAM cell is the Static Noise Margin (SNM), defined as the minimum DC noise voltage required to change the value stored into the cell. Traditionally SNM is represented graphically by using a "butterfly curve" made by voltage transfer characteristic (VTC) of one of the two inverters of the cell and the inverse VTC of the other inverter as shown in figure 1 where continuous lines show the initial curves. The SNM is visually obtained by the side of the largest possible square that can be drawn between two VTCs of the CMOS inverters. NBTI affects SNM by causing threshold voltage drift over time, thus lowering the static characteristics of the two inverters that form the 6T-SRAM cell. When the SNM of a cell falls below a threshold that allows safe storage of data then it can not be safely read or written.

SNM degradation strongly depends on the amount of stress time, however due to symmetric structure of SRAM cell, the value dependence is very weak and cell ages regardless of the value being stored. The worst case occurs when value stored in the cell does not change frequently and only one of the PMOS transistors degrades as shown in figure 1. Due to higher NBTI impact in this case, the SNM window will disappear faster and cell functionality will be lost earlier as compared to the case where both inverters exhibits same amount of degradation.

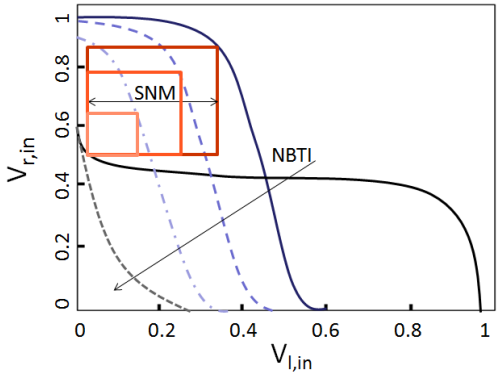

Fig. 1. Worst-case degradation of SNM as a result of V_{th} Drift

A. Related Work

Previous solution proposed to mitigate NBTI effects in SRAMs can be categorized into three types. One class of solutions include approaches that try to balance the degradation by equalizing cell value probabilities. The work of [7] proposed hardware and software schemes to periodically invert the entire contents of memory in order to equalize cell value probabilities. Another similar approach was proposed by [8] to invert the contents at word granularity along with much shorter inversion frequency.

A second set of solutions provided new cell designs with changes to mitigate aging effect. Approach of [9] introduced the design of customized NBTI-resilient cells consisting of a set of NAND gates to minimize the degradation ratio of all PMOS transistors in the cell. Another approach called "recovery boosting" [10] put both PMOS transistors of the cell into recovery mode which is done 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 idea of exploiting low-energy states to obtain aging reduction and therefore getting combined benefit of aging mitigation along with reduced energy consumption. The work of [11] evaluated the aging benefits obtained by the application of power gating to a memory cell and obtained much higher impact on aging as compared to controlling cell value probability. In [12], an architecture level solution acting on entire memory blocks was proposed which is based on power management solutions (DVS and power gating).

The work proposed in [13] introduced a time-varying cache indexing strategy called *dynamic indexing* in order to achieve a uniform distribution of idleness over the cache lines. This strategy guarantees the elimination of worst-case and so all leakage saving also maps in aging reduction. A coarse-grain version of this strategy was proposed in [14] which implements a uniform-size, multi-bank cache architecture to obtain uniform distribution among different partitions and thus achieve a better design point in aging/energy design space. The work of [3] used a different technique to deal with lines having worst-case idleness. The cache is split into uneven sized blocks with intention to hold lines with maximum accesses and thus shorter lifetime into a small memory block and discard only this block on first cell failure. In this way, the remaining cache will keep functioning with reduced efficiency.

III. Aging Aware Partitioning with Improved Performance

A. Dynamically Resizable Cache Architecture

The Dynamically Resizable Cache (DRC) [5] is an architecture originally devised for instruction caches in which the cache dynamically resizes itself to the size required during application execution and turns off the unused portion of the cache in order to suppress leakage energy.

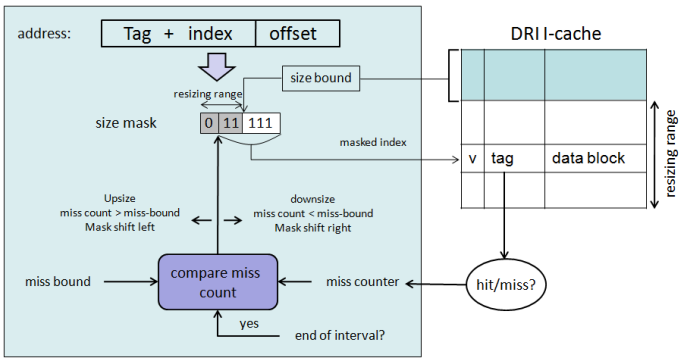

Fig. 2. Dynamically Resizable Cache Architecture [5].

In this architecture, the cache resizing process occurs in power-of-two: upon upsizing/downsizing, the cache size changes by a factor of two. Re-sizing the cache requires masking of the index bits of the address needed for a given cache size as shown in Figure 2. To monitor cache performance an application's execution time is divided into fixed-length intervals. At the end of each interval, miss count is compared to a preset value (miss bound) and cache size is determined accordingly. When downsizing (cache performance below threshold) the number of index bits is decreased by one (half the cache size); when upsizing (cache performance beyond threshold) the number of index bits is increased by one (double the cache size). Obviously, the increase/decrease in the number of index bits must be matched by a corresponding decrease/increase of the tag bits (not shown in the architecture) to guarantee that cache blocks can be retrieved. This implies some dynamic

enabling/disabling features of tag bits in the cache.
The main objective in the architecture of [5] is to reduce leakage energy which comes as a trade-off with cache performance. Next section will show how to exploit the DRC idea to reduce aging while preserving performance (which will ultimately provide energy reduction).

B. DRC for Aging Reduction

In our architecture, we have combined the concept of DRC along with cache partitioning to develop a new technique for improved cache performance along with reduction in both energy and aging. We used a simplified variant of the DRC in which *the cache is re-sized only once in its lifetime*, and only in one direction, i.e., downsized only. The resizing decision is taken based on the aging of the cache itself and it is not performance driven as in DRC. We do not need then to monitor application behavior nor we have to compensate by miss rate. Using DRC for aging reduction requires then a much simpler management. At the end of its normal lifetime, i.e., as soon as the first unit of access (a line) fails, instead of discarding whole cache, we only discard that specific portion (block) of the cache containing that line. This is the basic principle of the partitioned cache for aging, used in [3], [4]. The difference between the traditional partitioned architectures and our modified DRC lies in how the "dead" block is managed. In the partitioned approach the dead portion of the cache is marked as invalid, with the result that subsequent accesses to lines in that block will systematically result into a miss. This is why the approaches of [3], [4] suffer from a significant deterioration of performance after the first block fails.
In the proposed architecture, conversely, once a block becomes unusable, we *resize* the cache according to the DRC principle and then cache is flushed and re-configured to work as a smaller size cache. The benefits of this architecture for performance are evident. Since cache lines that age first are also the ones accessed the most, once the block containing lines that age quickly becomes unusable (because marked as invalid) will result in a large number of misses. Conversely, if we resize the cache, the application will just have a smaller (half the size) active set of lines.
An important consideration in this analysis is the fact that all cache lines will have degraded somehow (depending on the cache access pattern) during the first phase of their operation. In order to obtain the precise amount of aging it is therefore necessary to carefully calculate the degradation of each memory line during first phase and use this info as an initial level of aging for the second phase of cache operation.

C. DRC Architecture

Consider 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 consider the two halves of the cache as 2 blocks B_0 and B_1 of equal sizes though physically the cache structure is monolithic without any partition (Figure 3). The only feature that is partition-oriented is that the power management occurs at the block (half cache) granularity.

In order to accurately monitor aging, an aging sensor is required for each line. However due to our strategy discussed in section III-D, we only need to monitor the aging of second block B_1. We use an array of $L/2$ sensors, one for each line of

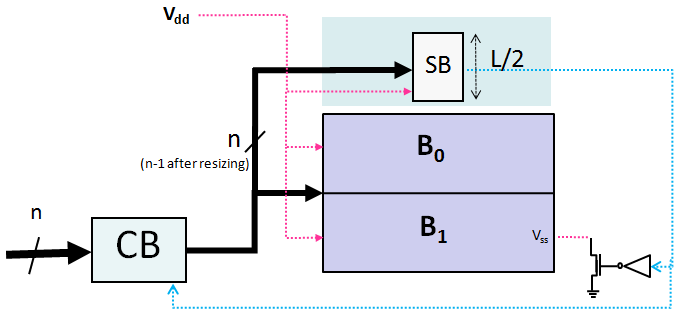

Fig. 3. Adaptation for Aging of a 2-Block DRC architecture.

the second block having same address as that of the cache line. In this way a sensor will be accessed for the same number of times as the cache line. Moreover, since we are interested to the aging of the whole block rather than that of single lines, all sensors outputs are OR-ed and therefore as soon as one of these sensors triggers indicating a faulty line, whole cache block will be disabled and cache will be reset to operate as half size cache (Figure 4).
We use power gating [15] to turn the unused block into a standby state (although any other implementation of the standby state is possible). Implementation of this technique can be seen in Figure 3 where we have used a sleep transistor. When a signal is received from the sensor block, the sleep transistor is turned off and thus all cells of the block are disconnected from ground. The block **SC** in the Figure 4

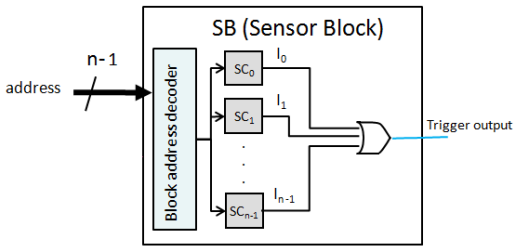

Fig. 4. Internals of the Sensor Block (SB).

denotes the aging sensor that monitors the aging of each individual line. A wide range of sensors are available in literature to track NBTI-induced aging but only few dedicated solutions are suitable to measure the aging of SRAM cells. Implementation of the sensor proposed in [6] perfectly fits our need as it is associated to a unit of access and therefore can easily be embedded into an existing memory array.
Finally, the control block (CB), upon receiving a signal from the sensors to indicate that a block has become unusable, is responsible for the masking of the address into an address of size $n - 1$.

D. Architectural variants

To keep things simple, we have considered a two partition case here where the lifetime of each block is determined by the line with least idleness. Moreover to avoid complications of address remapping in second phase, we only reuse the lower half of the cache for DRC method. However, it is obviously not possible in every case to have worst case idleness in second half and it can exist in either partition. If worst case idleness exist in lower partition then whole memory needs to be discarded when first block dies. To avoid this situation and get maximum advantage,

the knowledge of the idleness profile can be exploited. We have experimented with a selective swap strategy to move heavily accessed lines to second half of the cache which can easily be implemented by modifying the cache indexing function for a few, selected addresses. In this way we obtain reducible cache in all cases which provides significant improvement not only in miss rate but also in energy and aging results.

We have used a simple k-swap algorithm to repeatedly swap the address with minimum idleness in the first block with maximum idleness address of the second half. The number of swaps depends on the proportional benefit obtained by each swap and will be done only if beneficial in terms of average miss rate (AMR) and Effective lifetime (ELT).

The operation of k-swap algorithm is quite simple and is explained by the following pseudo-code which is self-explanatory: IL and IH are the idleness profiles of two cache partitions (lower half and higher half respectively).

1: k-Swap $(\mathbf{IL}, \mathbf{IH})$
2: **for** $l = 1 \ldots k$ **do**
3: $i \Leftarrow$ index of address with l-th minimum idleness in the first block.
4: j = index of address with l-th maximum idleness in the second block.
5: **if** $(\mathbf{IL}[i] < \mathbf{IH}[j])$ **then**
6: SWAP($\mathbf{IL}[i], \mathbf{IH}[j]$)
7: **end if**
8: **end for**
9: return

E. Metrics

In order to do a fair comparison against previous works, we need proper metrics for performance and aging. To this purpose, we utilize the concept presented in [3], [4] for *Average Miss Rate (AMR)*, which measures the performance of the cache by calculating average level of service offered over time and *Effective LifeTime (ELT)* defined as the product of lifetime and size of a memory block. It conceptually measures for how much time a memory block of a given size can be used.

Consider an idleness profile $\mathbf{I} = \{i_1, \ldots, i_L\}$ of a cache with L lines that can be partitioned into 2 blocks B_0 and B_1. The lifetime of a block is determined by the line with least idleness and the ELT can be expressed in this case as:

$$ELT = \frac{L}{2}(LT(min_0) + LT(min_1))$$

where min_i represents the line with minimum idleness of block i.

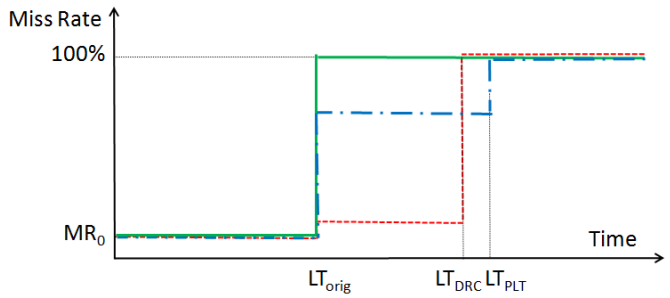

Fig. 5. Average Miss Rate.

AMR is an average metric, measured as the total number of misses over a reference time interval. The reference interval is the lifetime of the last-dying partition (B_0); since this values differs in general between PLT and DRC, we choose the largest of the two LT_{max}. Figure 5 shows an example of the AMR concept for three configurations: regular cache with no partitioning nor aging management (*Orig*), the non-uniform multi-bank architecture [3] *PLT*, and our proposed strategy *DRC*.

In regular cache, all L lines are usable reliably for an amount of time equal to LT_{orig}, when the first line dies. Being the cache monolithic, from that point on the cache experiences 100% miss rate (solid line).

In PLT and DRC approaches, one of the two blocks will have the same lifetime as original cache and will die at LT_{orig}, but the second half of the memory will keep functioning until LT_{PLT} and LT_{DRC} respectively. In the case of PLT, after the first half dies, miss rate degrades significantly due to repeated misses in the second half (dash-dot line). Conversely our DRC technique provides a more sophisticated solution: reconfiguration of cache will enable the access to frequent addresses through cache again and thus maintains the performance of the cache with negligible degradation (dashed line).

On the plot, AMR is equivalent to area below the curves divided by the lifetime LT_{max}, which in this case is LT_{PLT}. Obviously, smaller values of AMR are better and we can clearly see the advantage obtained through DRC by looking at red dotted line.

IV. Experimental Results

A. Experimental Setup

We have experimented our proposed architecture on a set of traces, extracted from the simulation of the MiBench suite [16] with an in-house cache simulator which utilizes aging models, derived from an industrial 45nm design kit provided by STMicroelectronics, to estimate aging. We define lifetime as the time after which the SNM of a cell 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.

Our target is application-specific systems which require one time analysis of idleness profile, extracted from cache simulator, to implement partitioning and selective swap strategy. It does not add any performance overhead during application execution. In fact, the only overhead is that of a sensor cell for each cache line which will result in less than 1% area overhead(1/128 using 16-byte lines as in our case) plus some wiring power overhead to connect various blocks.

B. Aging Results

Figures 6 and 7 show a comparison of the results obtained by our proposed architecture against previous work of [3] for an 8kB cache averaged over all benchmarks. The plot shows AMR and lifetime improvement over a regular, power-managed cache; with reference to [3] data refer to the case of $M = 2$ blocks.

Data is reported as a function of k as discussed in section III-D, where k denotes the number of swapped addresses. From figure 6 and 7, we can notice that the DRC strategy alone (even without swap, $k = 0$) improves AMR significantly and provides appreciable improvement in lifetime. AMR is only 8% in DRC as compared to 23% in case of PLT. Moreover, lifetime extension is almost equivalent to PLT in this case.

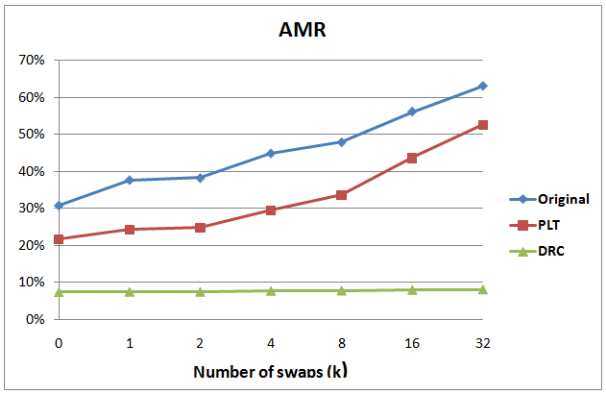

Fig. 6. Average miss rate of 8 KB Cache.

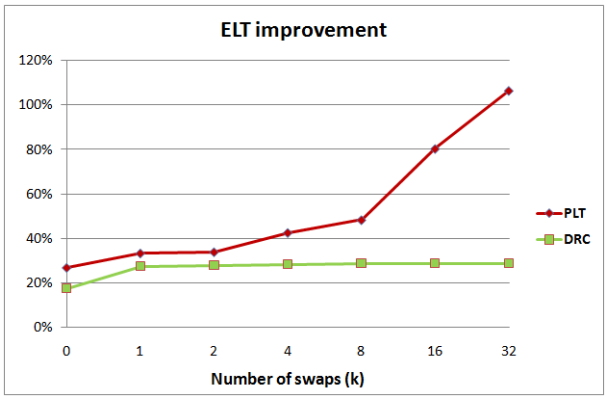

Fig. 7. Lifetime Improvement of 8 KB Cache.

Furthermore, we can also observe from these plots that in our strategy the AMR remains almost constant for all values of k while it grows quite rapidly in case of PLT due to the fact that lines with least idleness are the ones with maximum accesses. In other terms, the PLT architecture, in order to extend lifetime beyond the possibilities of DRC ($k \geq 4$), it has to heavily sacrifice performance. For instance, for $k = 16$ PLT achieves 80% ELT improvement, at the price of a 43% AMR.

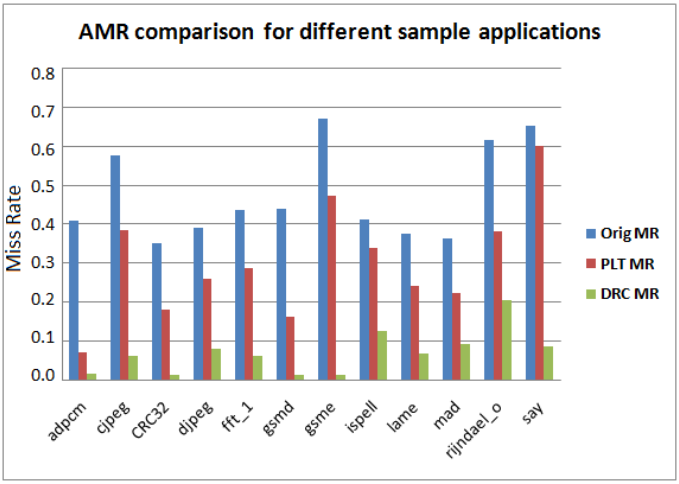

Fig. 8. Average miss rate of 8 KB Cache.

By analyzing the results obtained by selective swap strategy with different values of k, we see that even few swaps are sufficient to obtain the desired result, infact best trade-off is obtained for $k = 1$ and we will use this case in the rest of this section to obtain other results. We can better appreciate the benefit obtained through our technique by looking at figure 8 which shows a trace-by-trace result depicting AMR over the lifetime of a cache with 8KB cache size and $k = 1$.

Regarding energy reduction, our architecture is a modification of a traditional power-managed cache, so energy is saved due to the exploitation of idleness which is equivalent to energy saved by a conventional power-managed cache architecture without any aging management. The second component is due to the energy saved thanks to a reduced number of misses over time. It can be roughly quantified as the AMR after the death of the first line (LT_{orig} in Figure 5) multiplied by the cost of accessing the cache. This product accounts for the accesses to the next level of hierarchy that are avoided thanks to the fact that the cache is still active. The savings for this component are therefore highly correlated to the AMR figures.

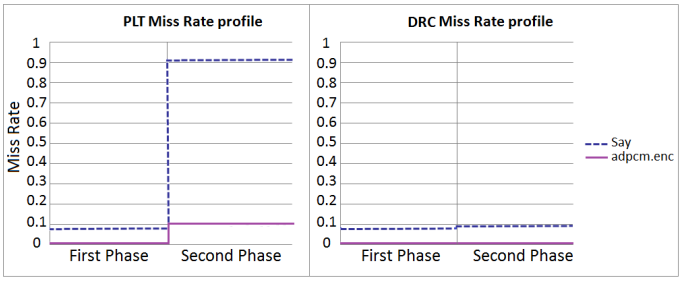

Fig. 9. Miss Rate Profile

To have a better view of the advantage obtained by our strategy, we have shown in figure 9 miss rate profile of two applications that demonstrates the change in miss rate when a partition dies. Selected applications show the extreme cases corresponding to minimum (`adpcm.enc`, about 10%) and maximum (`say`, about 90%) change in miss rate of an application after failing of the first line. We can clearly see that in case of DRC the change in miss rate can be significantly smoothed out thanks to the effective re-distribution of addresses resulting from cache resizing.

For the sake of completeness, we have also reported detailed results in table I and table II for the case of 8KB and 16KB cache (see next page). First three columns show a comparison of Average Miss rate among original, PLT and DRC architectures. Although there is variation across traces but the benefit in case of DRC is always sizable. Next three columns contain the absolute lifetime in all three cases and last two columns depicts the ELT improvement for PLT and DRC strategies over a simple power managed cache lifetime.

V. Conclusion

We have proposed a cache partitioning architecture that dynamically re-sizes and reconfigure the cache to provide a perfect solution for aging mitigation and improved miss rate with minimal hardware overhead. The proposed scheme exploits cache idleness in a smarter way to take advantage of those cache lines which are rarely used and can be re-utilized for concurrent aging and performance improvement. In addition to basic algorithm, we have exploited selective swap strategy that selectively swaps addresses among different blocks to get better aging and performance results.

By employing our DRC strategy, average miss rate of an 8KB cache reduces from 46% to 8% averaged over all benchmarks

TABLE I. DETAILED RESULTS FOR 8KB CACHE AND K=1

	Orig MR	PLT MR	DRC MR	Orig LT	PLT	DRC LT	PLT ELT Impr[%]	DRC ELT Impr[%]
adpcm.dec	0.409	0.071	0.015	3.39	5.65	5.59	33.33	32.34
adpcm.enc	0.359	0.046	0.011	4.96	6.89	7.65	19.49	27.18
cjpeg	0.576	0.383	0.061	8.19	18.36	12.95	62.13	29.07
CRC32	0.351	0.180	0.011	4.80	5.67	7.32	9.04	26.25
dijkstra	0.359	0.108	0.039	5.54	6.35	8.38	7.31	25.69
djpeg	0.389	0.259	0.081	10.32	13.35	15.62	14.66	25.64
fft_1	0.436	0.287	0.061	11.60	19.60	17.66	34.45	26.09
fft_2	0.390	0.268	0.066	12.79	19.92	19.45	27.85	26.02
gsmd	0.437	0.161	0.011	4.94	8.70	7.97	38.10	30.64
gsme	0.670	0.472	0.013	6.25	18.80	10.15	100.36	31.16
ispell	0.411	0.338	0.124	11.97	16.83	18.02	20.31	25.29
lame	0.374	0.240	0.066	16.24	21.02	24.60	14.71	25.75
mad	0.362	0.223	0.091	18.32	18.63	26.53	0.86	22.40
rijndael_i	0.565	0.383	0.200	8.11	15.74	12.79	47.05	28.90
rijndael_o	0.613	0.382	0.204	7.50	16.02	11.66	56.82	27.73
say	0.650	0.599	0.085	8.76	23.03	13.81	81.43	28.84
search	0.606	0.480	0.127	7.71	17.37	12.54	62.57	31.31
sha	0.339	0.106	0.010	8.53	8.53	12.82	0.00	25.19
tiff2bw	0.460	0.241	0.194	6.87	6.87	10.49	0.03	26.41
Average	**0.461**	**0.275**	**0.077**				**33.18**	**27.47**

TABLE II. DETAILED RESULTS FOR 16KB CACHE AND K=1

	Orig MR	PLT MR	DRC MR	Orig LT	PLT	DRC LT	PLT ELT Impr[%]	DRC ELT Impr[%]
adpcm.dec	0.468	0.063	0.007	3.62	6.81	5.85	43.99	30.72
adpcm.enc	0.354	0.034	0.004	5.63	8.24	8.72	23.18	27.44
cjpeg	0.572	0.433	0.057	9.11	20.40	13.89	61.97	26.24
CRC32	0.347	0.145	0.011	5.47	6.52	8.30	9.61	25.91
dijkstra	0.390	0.136	0.034	5.42	7.65	8.71	20.62	30.39
djpeg	0.384	0.222	0.071	11.12	14.91	16.97	17.02	26.30
fft_1	0.418	0.243	0.045	12.16	20.48	18.64	34.17	26.60
fft_2	0.362	0.188	0.046	13.97	21.45	20.94	26.78	24.95
gsmd	0.444	0.322	0.009	5.54	9.94	8.63	39.63	27.81
gsme	0.657	0.272	0.012	6.29	18.23	10.13	94.85	30.49
ispell	0.443	0.322	0.112	12.45	20.31	19.11	31.60	26.77
lame	0.353	0.176	0.057	17.40	18.26	25.86	2.47	24.32
mad	0.364	0.205	0.086	19.63	20.08	28.73	1.14	23.18
rijndael_i	0.494	0.251	0.168	9.16	16.15	13.88	38.13	25.74
rijndael_o	0.578	0.428	0.173	8.42	17.81	12.83	55.73	26.15
say	0.405	0.139	0.064	8.42	10.66	13.43	13.36	29.81
search	0.631	0.517	0.126	7.68	18.54	12.30	70.69	30.04
sha	0.330	0.056	0.007	8.48	8.48	12.62	0.00	24.45
tiff2bw	0.399	0.201	0.121	5.91	5.92	9.25	0.04	28.24
Average	**0.442**	**0.229**	**0.064**				**30.79**	**27.13**

in contrast to PLT where AMR goes down to 28% only. A similar trend is shown by 16KB cache.

REFERENCES

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

[2] 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.

[3] H. Mahmood, M. Loghi, E. Macii, M. Poncino, "Application-Specific Memory Partitioning for Joint Energy and Lifetime Optimization", *DATE'12:Design, Automation and Test in Europe*, March 2012, pp. 364-369.

[4] H. Mahmood, M. Loghi, E. Macii, M. Poncino, "Aging-aware caches with graceful degradation of performance", *VLSI-SoC'12: IEEE/IFIP 20th International Conference on VLSI and System-on-Chip*, vol., no., pp.237,242, 7-10 Oct. 2012

[5] Y. Wang et al., "An energy-efficient high performance deep submicron instruction cache." *IEEE Transactions on VLSI, Special Issue on Low Power Electronics and Design* (2001)

[6] Qi, J., J. Wang, B. H. Calhoun, M. Stan "SRAM-based NBTI/PBTI sensor system design," *DAC-47 : 47th Design Automation Conference*, June 2010, pp. 48.1–48.4.

[7] 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.

[8] 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.

[9] T. Siddiqua, S. Gurumurthi, "Recovery Boosting: A Technique to Enhance NBTI Recovery in SRAM Arrays," *ISVLSI'10: IEEE Annual Symposium on VLSI*, July 2010.

[10] 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.

[11] A. Calimera, M. Loghi, E. Macii, M. Poncino, "Analysis of NBTI-induced SNM degradation in power-gated SRAM cells," *ISCAS'10: International Symposium on Circuits and Systems*, pp. 785–788, May 2010.

[12] 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.

[13] 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

[14] 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.

[15] Powell, M.; Se-Hyun Yang; Falsafi, B.; Roy, K.; Vijaykumar, T.N., "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.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", *IEEE 4th Annual Workshop on Workload Characterization*, pp. 3–14, Dec. 2001.