ABSTRACT
This work addresses the aging of the memory sub-system due to NBTI (Negative Bias Temperature Instability) in systems that have to provide a guaranteed level of service, and specifically, a guaranteed lifetime. Our approach leverages a novel cache architecture in which a smart joint use of redundancy and power management allows us to obtain caches that meet a desired lifetime target with minimal energy consumption. This is made possible by exploiting the possibility of putting the cache sub-block used as for redundancy into a deep low-power state, thus allowing more energy saving than a regular architecture. Sacrificing a portion of the cache for aging mitigation only marginally affects performance thanks to the non-linear dependency of miss rate versus cache size, which allows to find the best cache size that maximizes the objective. Simulation results show that it is possible to meet the target lifetime by achieving energy reductions (measured over the lifetime of the system) ranging from 3X to 10X (2X to 8X) for a lifetime target of 15 (25) years, with marginal miss rate overhead.

1. INTRODUCTION
Many mission-critical applications require guaranteed lifetime of their operations, and therefore of the hardware implementing their functionality. In some cases, a guaranteed level of service must also be granted, for instance in terms of throughput. Such constraints are usually enforced by means of various reliability-enhancing solutions mostly based on redundancy [1], which are typically not energy-friendly, because the replicas consume extra energy (even when power managed) and so does the control logic required for managing the operations.

In this work we focus on a particular type of reliability threat, namely the aging of CMOS devices, and in particular the one due to NBTI (Negative Bias Temperature Instability), which is regarded as the most significant source of device aging in nano-scale technologies [2]. More specifically, we target the aging of SRAMs in the memory sub-system, which is the most critical component for warranting reliable and timely operations.

Aging induced by NBTI is a value-dependent effect: when a pMOS device is under negative bias (a logic “0” applied to its gate terminal), this causes the threshold voltage (and propagation delay) to increase over time. Moreover, NBTI is partially reversible: the application of a logic “1” to the gate terminal will cause the device to partially recover its threshold voltage. Therefore, for generic circuits it is essential to evaluate the distribution of “0”s and “1”s to determine the actual aging. In SRAMs, conversely, the value dependence of NBTI is weaker than in generic circuits: given the symmetric structure of a SRAM cell there is in practice no recovery effect and the cells age regardless of the stored value.

Previous works have observed that there exists a correlation between the concept of idleness exploited in power management strategies and aging. In particular, the “hardware” implementation of a low-power state (i.e., voltage scaling for dynamic power and power/ground gating for static power) can be leveraged to reduce NBTI-induced aging [4, 5]. Therefore, several works have properly revisited various traditional power-managed cache and memory architectures under an aging-related metric so to achieve concurrent energy and aging improvements [12, 13].

All these approaches are based on the idea of transforming the idleness resulting from a given workload (which is exploited to reduce energy) into an equivalent benefit for aging as well. Direct use of the idleness is not always possible: while for energy it is the total idleness that matters, for aging (a worst-case metric) it is the distribution of the idleness that matters. Such a transformation consists either of a proper arrangement of addressing mechanisms ([12]) or of multi-banked organization of an SRAM ([13]). None of these approaches however relies on widely used reliability-enhancing paradigm, that is, the use of redundancy.

In this work we show that, by properly combining the two above approaches and redundancy, it is possible to push the energy reduction beyond the limits of previous works. The rationale of our architecture is to use sub-banking not to reduce the impact of the worst-case idleness ([13]) but rather as an extra memory space over which better distribute idleness. In other terms, we only use a subset of the cache to store values. Energy reduction is achieved because the cache sub-block used for redundancy can in be put into a non state-preserving state during standby without compromising performance. As shown in [12], this is not feasible if the whole cache is active, due to the poor exploitability of a non state-preserving state.

The use of a virtually smaller cache, however, affects performance. We show that this impact is marginal, thanks to the non-linear dependence between miss rate and cache size, which saturates for typical cache sizes.

The proposed approach allows, for a given lifetime target, to significantly improving the overall energy consumption of the cache (measured over the target lifetime), with truly marginal degradation of miss rate.

2. BACKGROUND AND RELATED WORK
2.1 Background
For space limits we limit ourselves to a summary of the effects of NBTI-induced aging in SRAM cells. For details about the modeling of NBTI effects and its dependence on
technological and environmental parameters, we refer the reader to the classical papers on the subject (e.g., [2, 3]). The threshold voltage drift caused by NBTI does not truly affect the delay of a SRAM cell but rather it decreases its Static Noise Margin (SNM), i.e., the minimum DC noise voltage required to change the state of the cell. This affects the stability of a cell, because if the SNM of a cell falls below some technology-dependent threshold, the data stored in the cell cannot be safely read or written. Another characteristics of NBTI effects on SRAM cells is the weak dependence on logic values: whatever the values stored, the cell will age anyway. Distribution of the values can however impact the cell stability: the best-case occurs when both inverters in the cell exhibit the same amount of degradation, i.e., when a cell stores a 0 and a 1 with equal probability [6].

Based on this considerations and the technology used for our simulations, we define lifetime of a SRAM cell as the time after which its SNM has decreased by 20%.

2.2 Related Work

Solutions proposed to mitigate NBTI effects in SRAMs fall in three main categories. The first class includes methods that try to equalize cell value probabilities using various hardware and software strategies [6, 7].

Another type of approach aims at designing customized NBTI-resilient cells [8, 9]. In [8] a new NAND-based cell structure is proposed so that minimum degradation ratio for all PMOS transistors in the cell is obtained. Another solution called recovery boosting [9] allows both PMOS devices in the cell to be put into the recovery mode by raising the ground voltage and bit-lines 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 [13, 12]. Assessment at the architectural level on entire memory blocks of power management solutions (based on both DVS and power gating) were evaluated in [13].

The work of [12] 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.

One architectural approach based on redundancy is the one proposed in [14], in which a spare cache sub-array is used to replace, on a rotating basis, selected sub-arrays with excessive aging, which are then put into a special wearout-recovery state implemented through a sort of power gating.

This approach is more fine-grain than ours, but requires considerable overhead. The “scheduling” of the spare unit must be explicitly managed, and the choice of the unit to be recovered requires per-unit detection of the level of aging. Conversely, our scheme is suitable also for small-scale systems and does not require modification on the internals of the cache.

3. MOTIVATION AND CONCEPT

Consider a system with a cache (for simplicity, direct-mapped with a pre-defined line size) of a given size $S$, measured in cache lines, determined according to some architectural considerations. This baseline cache will have a given power consumption, lifetime, and performance (e.g., miss rate). Since all these quantities are affected by cache size, we denote them by $P(S)$, $LT(S)$, $MR(S)$. Suppose also that the system constraints include the warranty that the system (in this case, the cache) guarantees operation for a specified target lifetime3. Our objective is to devise an organization the cache sub-system so that the lifetime target is met by using the least possible power and with smallest possible performance penalty.

One way to approach this problem is to try to extend the default lifetime by using the intrinsic idleness of the application that uses the cache ([12, 13]). For example the “dynamic indexing” proposed in [12] proposes a simple and effective solution that exploit 100% the available idleness by distributing it over the cache lines so that all lines exhibit the same idleness (and lifetime). This solution maximizes lifetime, but the fact that all the cache lines are used limits the possibility of power managing the lines. As a matter of fact, [12] ruled out a low-power state based on power gating (which provided longer lifetimes) because it is less exploitable from the energy standpoint due to much longer break-even times caused by miss penalty.

Our method overcomes this limits as follows. We keep only a subset $S' < S$ of the cache lines as “active” cache lines; The remaining $S - S'$ act as “spare” lines that can be used to mitigate the aging of the whole cache. The benefit from the energy standpoint is intuitive: the “inactive” portion of the cache can be put in a non state-preserving low-power state (thus saving more energy). However, this also extends lifetime because aging is virtually removed under footer-based power gating (actually there is a recovery)/[12], and the “inactive” part of the cache is less aged when it gets reused.

On the other hand, under this scheme we are virtually using a smaller cache, so performance can be impacted. However, thanks to the dependence of the the three metrics $(E(), LT(),$ and $MR())$ versus cache size, it is possible to make this performance overhead negligible.

One final note concerns the concept of “portion”. In order to make addressing simple, the above mentioned portions must coincide with power-of-two fractions of the initial cache size. In the rest of the section we illustrate the basic dependencies of the three above metrics versus cache size. In Section 4, we will then discuss architectural issues and the optimization strategy.

3.1 Models

In the following, we consider a power-managed cache in which a cache line is the atomic unit of power management.

3.1.1 Lifetime

By running a given workload with a cache simulator we extract then the exploitable idleness (i.e., idle intervals longer than some break-even time) of each line; the latter is a metric of how much energy can be saved. Using the characterization methodology described in [5], it is possible to extract the lifetime (as defined in Section 2) of an SRAM cell as a function of its idleness, that is, the percentage of time in which it is in standby state. The lifetime of a cell determines the lifetime of the line it belongs to, for a line is the atomic unit of access of a cache. The lifetime of the entire cache is the one of the earliest failing line (i.e., the line with smallest idleness).

As already mentioned, special architectural arrangement such as the dynamic indexing in [12] can distribute the idleness over the cache lines uniformly so that

3 We are assuming the regular lifetime of the cache is shorter than the target.
all lines fail at the same time (and worst-case coincides with average case).

Figure 1 shows the result of the characterization for a dynamically indexed 32kB cache. The plot refers to a DVS-based implementation of the standby state ("drowsy"), which is the mechanism used for the "active" portion of the cache in our redundant architecture. Conversely, the "inactive" portion of the cache can be power-gated without incurring the cost of a miss penalty, since it is not used. For the inactive block, idleness is 100% and aging is zero.

Figure 1: Lifetime of SRAM cell vs. % Idleness.

Lifetime is obviously monotonically increasing with respect to the percentage idleness; the intercept on the Y-axis (3.96 years) denotes the baseline lifetime of the cache (0% idleness). Conversely, 100% idleness (a theoretical value) implies that the all cache lines are always off; the corresponding lifetime value (about 12 years) represents the intrinsic benefit achieved by DVS. The fact that this value is not infinity (as it would be in the case power gating is used [12]) is because DVS does not nullify aging but just mitigates it. The figure also shows the interpolation to be used in our analytical formulation. We found that the cubic function

\[
LT(I) = 3.96 + 4 \cdot I - 7.0 \cdot I^2 + 11.1 I^3
\]  

well approximates the simulation data (average error = 0.18%, maximum error = +1.97%).

In order to express lifetime in terms of cache size, we need first to establish a relation \(I(S)\) between idleness and cache size. Intuitively, a larger cache will have higher average idleness (same accesses with more targets available), so \(I(S)\) will be a monotonically increasing function. The actual behavior has to be characterized by running a cache simulator under different workloads and for different cache sizes. Figure 2 shows the result of the characterization.

As one would expect, the idleness is very low for very small cache sizes, then it grows quickly and tends to saturate. Conceptually, idleness becomes 100% for infinite caches sizes. We fitted the curve to a template \(f(x) = 1 - \frac{1}{ax+b} \). A good fit is obtained with

\[
I(S) = 1 - \frac{1}{0.005 \cdot S + 0.92}
\]  

For caches sizes larger than 32 bytes, the average (maximum) error is 5.6% (8.3%).

By composing the two functions, we can finally get \(LT(S)\) by replacing Equation 2 into Equation 1. Due to the shape of the two curves, the results \(LT(S)\) curve roughly follows the shape of the \(I(S)\) curve: lifetime has a steep increase for intermediate cache sizes (between 1K and 4K) and then it quickly saturates.

3.1.2 Miss Rate

There are many empirical studies that have tried to correlate cache miss rate versus its size ([15, 16]). A widely accepted model that is relatively accurate over various cache organizations and different workloads states that miss rate roughly goes as \(\sqrt{n}\). We have therefore characterized our embedded applications to extract the actual dependency and have fitted the results to a template \(f(x) = \frac{3.6}{\sqrt{0.45 \cdot S + 30}}\)

obtaining the following function:

\[
MR(S) = \frac{3.6}{\sqrt{0.45 \cdot S + 30}} \]  

Figure 3 pictorially shows the simulated data and the fitted curve of Equation 3.

Figure 3: Miss Rate as a Function of Cache Size.

If we consider the complement curve of the hit rate, we could recognize some similitude between the \(LT(S)\) and \(MR(S)\) curves. This is intuitive since idleness and miss rate are somehow correlated: remember that we are considering average idleness. For instance, if idleness is low (as in small caches), very likely the miss rate will also be high. However, the correlation gets weaker as cache sizes increases: idleness has to do with the distribution of accesses, whereas miss rate considers the specific location of accesses. Therefore, we expect our strategy providing more benefits for larger caches.

3.1.3 Power

A line consumes static power only for the fraction of time it is used, that is, \(1 - I(S)\). Hence, the static power spent by a cache in "drowsy" mode is:

\[
P_{\text{static}}(S) = S \cdot (1 - I(S)) \cdot P'(S) + S \cdot P_{\text{idle}}(S)
\]
where \( P' \) is the leakage normalized with respect to line size, and \( P_{idle} \) is the (normalized) leakage in the idle state implemented using DVS. In our technology, \( P_{idle} \) is not zero and is about one order of magnitude smaller than \( P' \).

Notice that the model explicitly exposes the dependence of both \( P' \) and \( P_{idle} \) (which is a fraction of \( P' \) ) on \( S \). This dependence accounts for the fact that a given cache size implies a specific aspect ratio (rows and columns) of the tags and data arrays, which affects the capacities of the interconnects (bit- and word-lines) as well as the structure of the decoder, the number of MUXes and sense amps. It is therefore essential not to consider \( P' \) and \( P_{idle} \) not as constant values.

Dynamic power, conversely, is not affected by the nature of the idle state: it consists of the sum of the power spent during cache accesses plus the power spent by the background memory when a miss occurs. Since the cache is used at each cycle (at least for the instruction fetch), the dynamic power can be expressed as:

\[
P_{dynamic}(S) = P_{cache}(S) + MR(S) \cdot P_{miss}
\]

Where \( P_{cache} \) is the power spent for accessing the cache, and \( P_{miss} \) is the power spent by the lower levels of the memory hierarchy when accessed. Therefore, while exploiting idleness is beneficial on controlling static power, at the same time the miss rate must be kept under control, in order to avoid an increase of dynamic power.

However, when we move to considering energy, and thus include the temporal dimension in the analysis, the choice of the lifetime target significantly impacts the overall energy consumption.

Assume a lifetime target \( T \) and that the cache becomes unusable at some time \( LT_{cache} < T \). It is clear that after time \( T \), the miss rate jumps at 100% (no cache), and each memory access will imply access to a background memory. Although the cache, once it has become unusable, can now be put into a very low-leakage, the relative benefit is a marginal compensation against the huge increase dynamic power consumption.

We can express the total energy spent by the memory during the system operating time \( T \) , can be expressed as:

\[
E_{total} = LT_{cache} \cdot (P_{static} + P_{dynamic}) + (T - LT_{cache}) \cdot P_{miss}
\]

Since \( P_{miss} \) is about two orders of magnitude larger than the power spent by the cache, it is evident that a long lasting cache (i.e., a cache with a larger \( LT_{cache} \) ) will provide larger energy benefits (or, in other words, will be an energy effective solution for a larger time).

### 4. ARCHITECTURE AND EXPLORATION STRATEGY

Consider a direct-mapped cache with \( S = 2^n \) lines where \( n \) is the number of the index bits of the cache address. As discussed in Section 3.1.1, we assume the cache line is the atomic unit of power management. Figure 4-(a) depicts the conventional scenario resulting from the use of Dynamic Indexing (DI) [12]: the original idleness distribution (shown on the left) is transformed by the DI block into a uniform distribution, thus making worst and average cases to coincide. This is achieved by changing the indexing over time; the change is triggered by an Update signal (refer to [12] for more details). Let \( I_{avg,k} \) be this average idleness. In this scenario, the entire cache is kept active (blue color), that is, any line is addressable at any time. Clearly, at a given time, some lines will be in a standby state; in this case this state must be implemented through DVS (state-preserving) to avoid the miss penalty in the re-activation.

![Figure 4: Traditional Dynamic-Indexing Operations (a) and Proposed Architecture (b).](image)

Figure 4-(b) depicts the proposed scenario. The cache is conceptually split into \( k \) portions (\( S' = \frac{S}{k}, k = 2 \) in figure), only one of which is active at a given time. This implies that the system effectively uses a smaller cache. The intrinsic idleness (the distribution shown on the left) is now different from the previous case and its average will also be smaller: intuitively, \( I_{avg,k} \approx \frac{I_{avg,1}}{k} \), because the same accesses are now distributed over a portion of the address space. Dynamic indexing will again flatten the distribution and make it uniform. Therefore, all lines in the active block will degrade of the same amount.

While a cache portion is accessed, the other ones (in gray) can be put in a non state-preserving state implemented by a footer-based power gating. This state induces and electrical state inside each cell where all nodes drift towards a logic “1”, which is the recovery state for NBTI [5]. While this very deep standby state cannot be used for the active block, we can afford it for the inactive ones, which can easily lose their values without impact on performance.

Whenever the Update signal is triggered to change the indexing mechanism, the selector brings up another portion of the cache (now rejuvenated by having passed some time in the recovery state) and puts the first one into the non-preserving standby state. As in the conventional architecture, updating the indexing needs a flush of the entire cache: this is not a true overhead because updates can be synchronized to architectural events that require cache flushes (e.g., context switches). The overall effect is a sort of bi-dimensional and heterogeneous indexing over the cache blocks. Each one will age of the same amount, but when not used they save much more energy than in the regular case.
One relevant observation that is worth mentioning is that using one less bit for the index to address a smaller cache \((n-1\) instead of \(n\)) implies storing one extra tag bit; the total width of the address is clearly fixed. This is typical in other variants of variable-size caches, like for instance the Dynamically Resizable I-cache [17]. The impact of such a larger tag array, however, will be considered by the accurate explorative analysis below.

Given the above architecture, the objective is to determine what cache size, and what value of \(k\) yields the least energy for a given target lifetime. This problem can be viewed as a novel, aging-aware optimization scenario in which lifetime becomes an explicit constraint (as opposed to a traditional performance metrics) and energy as the variable to be optimized. This has also impacts on how we evaluate energy.

Finding an analytical solution to this optimization problem by using the models of Section 3.1 is in theory possible. However, two consideration suggest a much simpler approach. First, the solution would require complex mathematical solutions due to the non strong linearity and the variety of the functions describing the models. Second, and more relevant, the design space is very sparse: not only the main variable \(S\) is discretized, but it can assume only a very limited number of values (the mpowers of two). Since we assume \(S=1\text{KB}\) as the atomic unit of instantiation, we need to evaluate less than 10 solutions. For this reason, our analysis is based on an explorative approach in which the cache size parameter is swept from the minimum to the maximum value and the values of the other metrics are evaluated.

5. SIMULATION RESULTS

5.1 Experimental Setup

We have assessed the proposed architecture by evaluating a set of applications from the MediaBench suite [18]. We implemented a cache simulator that evaluates the miss rate and, for each cache access, estimates the power consumption, leveraging power models derived from an industrial 45nm design kit provided by STMicroelectronics. The power models also account for the leakage power spent by the cache lines (in the active and in the low-leakage “drowsy” state). Furthermore, for each cache miss, the simulator accounts for the energy spent in the background memory to refill the target line. Finally, the simulator keeps track of the useful idleness of each line, in order to evaluate the cache lifetime as depicted in Section 3.1.1.

The following tables report the average power of the whole memory hierarchy (cache and background memory), both static and dynamic, normalized to the power required to access a 8kB cache. Lifetime values (defined as in Section 2) are expressed in years, while miss rates are percentages.

In our analysis, we considered the 1kB cache as the smallest unit that can instantiated, since the preliminary considerations drawn in Section 3.1 indicate that too small caches incur in too large overhead to benefit from our strategy.

5.2 Power and Lifetime Results

In the first experiment, we compared the non-redundant, power managed caches, with and without dynamic re-indexing, against the proposed redundant architectures where the memory array is split in two or in four blocks. The various configurations are evaluated in terms of power consumption and lifetime.

<table>
<thead>
<tr>
<th>Cache</th>
<th>No redundancy</th>
<th>2-blocks</th>
<th>4-blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>(P_{\text{orig}})</td>
<td>(LT_{\text{orig}})</td>
<td>(LT_{\text{dyn}})</td>
</tr>
<tr>
<td>1 kB</td>
<td>10.12</td>
<td>5.42</td>
<td>8.87</td>
</tr>
<tr>
<td>2 kB</td>
<td>7.63</td>
<td>5.90</td>
<td>10.53</td>
</tr>
<tr>
<td>4 kB</td>
<td>5.76</td>
<td>6.05</td>
<td>11.29</td>
</tr>
<tr>
<td>8 kB</td>
<td>5.65</td>
<td>4.34</td>
<td>6.16</td>
</tr>
<tr>
<td>16 kB</td>
<td>5.21</td>
<td>7.63</td>
<td>6.34</td>
</tr>
<tr>
<td>32 kB</td>
<td>2.66</td>
<td>2.66</td>
<td>6.83</td>
</tr>
</tbody>
</table>

Table 1: Total Power (Normalized) of the Memory Hierarchy and Cache Lifetime.

Results reported in Table 1 show that while the dynamic re-indexing allows a significant extension of the cache lifetime, still cannot guarantee much more than about 12 years of cache operating time (for the largest cache considered), that is quite near to the upper bound shown in Figure 1. If the required lifetime is longer, the redundant architecture can break this barrier and offer a substantial improvement. A 2-block architecture, can reach about 24 years of lifetime, and extending to a 4-block cache a time horizon longer than 40 years, even for relatively small caches.

Power consumption is clearly larger in redundant architectures; using a smaller “active” set of memory locations implies a larger miss rate. If we measure power over the actual target lifetime of the system, the power balance (initially in favor of the regular architecture) eventually becomes in favor of the redundant one. In other words, there is a power penalty only until the time during which the non-redundant cache is working; after it becomes unusable, the redundant cache becomes also power efficient. This effect can be noticed in Tables 2 and 3, reporting power figures for a \(T = 15\) and \(T = 20\), respectively.

<table>
<thead>
<tr>
<th>Size</th>
<th>(P_{\text{orig}})</th>
<th>(P_{\text{dyn}})</th>
<th>(MR)</th>
<th>2-blocks</th>
<th>4-blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 kB</td>
<td>37.78</td>
<td>27.81</td>
<td>51.18</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2 kB</td>
<td>35.41</td>
<td>21.28</td>
<td>38.89</td>
<td>10.12</td>
<td>17.48</td>
</tr>
<tr>
<td>4 kB</td>
<td>34.22</td>
<td>17.57</td>
<td>31.80</td>
<td>7.63</td>
<td>12.96</td>
</tr>
<tr>
<td>8 kB</td>
<td>33.27</td>
<td>15.13</td>
<td>27.10</td>
<td>5.76</td>
<td>9.36</td>
</tr>
<tr>
<td>16 kB</td>
<td>32.20</td>
<td>13.56</td>
<td>24.02</td>
<td>4.34</td>
<td>6.57</td>
</tr>
<tr>
<td>32 kB</td>
<td>30.30</td>
<td>12.82</td>
<td>22.43</td>
<td>3.21</td>
<td>4.29</td>
</tr>
</tbody>
</table>

Table 2: Total Power (Normalized) of the Memory Hierarchy and Average Miss Rate for \(T = 15\).
Table 3: Total Power (Normalized) of the Memory Hierarchy and Average Miss Rate for $T = 25$.

<table>
<thead>
<tr>
<th>Size</th>
<th>$P_{orig}$</th>
<th>$P_{dyn}$</th>
<th>MR</th>
<th>$P$</th>
<th>MR</th>
<th>$P$</th>
<th>MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 kB</td>
<td>44.04</td>
<td>38.06</td>
<td>70.71</td>
<td>22.69</td>
<td>14.42</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2 kB</td>
<td>42.62</td>
<td>34.14</td>
<td>63.34</td>
<td>22.69</td>
<td>14.42</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4 kB</td>
<td>41.91</td>
<td>31.91</td>
<td>59.08</td>
<td>14.85</td>
<td>26.67</td>
<td>10.12</td>
<td>17.48</td>
</tr>
<tr>
<td>8 kB</td>
<td>41.34</td>
<td>30.45</td>
<td>56.26</td>
<td>10.39</td>
<td>18.17</td>
<td>7.63</td>
<td>12.96</td>
</tr>
<tr>
<td>16 kB</td>
<td>40.69</td>
<td>29.51</td>
<td>54.41</td>
<td>7.47</td>
<td>12.52</td>
<td>5.76</td>
<td>9.36</td>
</tr>
<tr>
<td>32 kB</td>
<td>39.56</td>
<td>29.07</td>
<td>53.46</td>
<td>5.59</td>
<td>8.83</td>
<td>4.34</td>
<td>6.57</td>
</tr>
</tbody>
</table>

6. CONCLUSIONS

We have proposed a redundant cache architecture that allows to achieve energy-optimal cache configurations while meeting a given lifetime target. The proposed scheme allows to overcome the limit resulting from non-redundant schemes that are based on redistribution of addresses, and exploits the fact that the drawbacks resulting from effectively using a smaller cache (higher power and higher miss rate) are respectively transformed into a benefit (power) or made negligible (miss rate). For large lifetime targets (15 and 25 years), our redundant architectures can reduce the energy consumed by the memory hierarchy by a factor from 3x to 10x (for 15 years), and from 2x to 8x (for 25 years).

7. REFERENCES