# CMP SCI 635: Modern Computer Architecture

## A Proactive Wearout Recovery Approach for Exploiting Microarchitectural Redundancy to Extend Cache SRAM Lifetime

Name: Kunjal Panchal  Date: 19th Sept, 2019

Student ID: 32126469  Paper: 5 – Wearout [Shin08]

Strengths:

1. The proactive strategy doesn't introduce any extra monetary costs as the number of components will be as many as there are in reactive strategy. It just uses each component from the beginning, instead of waiting for one component to get out of order and then switching it with the exact same inactive component. This makes sure that the component wearout occurs way after the expected time, if we were using the proactive strategy, as the active component in the brink of wearout will be switched with another inactive one which is fresh to use, in a rotational basis. This allows components to recover, while reactive strategy can't recover the already failed modules. This extends the lifetime of that component.

2. The way 6T SRAM is implemented to proactively avoid the wearout is also quite simple and efficient as there are power rails already available to keep $V_{dd}$ at a level; and while one SRAM becomes inactive, the next one will be active through cross switching inverters, which also doesn't require much extra circuitry to add. This implementation of wearout recovery mode costs negligible additional area overhead.

3. Graceful performance degradation improves the lifetime reliability of the cache SRAM by about 10% to 27%, but suffers a performance loss of 6% to 13%. ECC and the proactive approach suffers a less than 1% performance loss while providing lifetime reliability enhancement of up to about a factor of three and seven, respectively.

Weaknesses:

1. Invalidating the cache before draining it looks like it will have a lot overhead because of all the cache misses suddenly occurring. Migration method won't change the state of the cache lines, but it can't migrate the lines which requires data array access such as reads and writes, those are bound to get rejected, resulting in cache misses, and thus, overhead.
2. Array $a_0$is always in rotation, active. The paper doesn't explain what happens if $a_0$fails. If that array failed permanently, no proactive wearout recovery will possible.
3. Accesses to arrays entering into recovery mode are delayed throughout the drain process. There may also be additional delay due to contention on cache resources such as cache read/write ports, cache bus bandwidth, and/or write-back related queues and logic to implement the invalidations or migrations. In addition to these delays, extra cache misses occur if invalidated lines are accessed after the drain process completes.

4. The migration drain mechanism with dedicated links requires additional wiring between adjacent arrays and another level of multiplexing at write port data inputs. The fewer the number of bits read from an array, the smaller the overhead of implementing the links. However, lower bandwidth requires more cycles to transfer data between arrays, causing additional latency for the drain process.

Questions/Assertions:

1. "*We also assume that the recovery mode of the arrays is scheduled as a recurring sequence of two equal intervals or set of sequences of intervals of diminishing duration for the two PFETs of the array cells.*" Does the hardware that keeps track of voltage levels and other stats which helps in determining the initial interval and then the diminishing durations, introduce much overhead?
2. What happens after one or more array wears out? How does the rotation logic omit those arrays from being considered as "active"? How to keep track of the failed and runnable arrays?
3. Is the idea of a combination of proactive and reactive mode of wearout recovery proposed? How about arrays working in proactive mode but along with that there are few more redundant arrays which will not be activated till all arrays of the rotational cycle are completely worn out? Is the idea worth exploring? Or a situation which might demand this will never arise? {Also stated in the last section, the both techniques are orthogonal.}
4. A paper on "*Estimation of Remaining Life Using Embedded SRAM for Wearout Parameter Extraction*" [link: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7184952] explores how to estimate the remaining lifetime of a circuit that incorporates both random logic and memory using the failed memory bits as a sensor. The number of failed memory bits correlates with the remaining lifetime, but varies as a function of wearout mechanism and use conditions. Hence, the proposed work also incorporates algorithms and test methodologies to diagnose the cause of failure. By linking the failed bits to the wearout mechanism and by using lifetime simulation, the remaining lifetime can be estimated.

   This might fit well with proactive wearout checking proposed in the paper we are discussing.