2016 IEEE International Parallel and Distributed Processing Symposium

# Re-NUCA: A Practical NUCA Architecture for ReRAM based last-level caches

Jagadish B. Kotra, Mohammad Arjomand, Diana Guttman, Mahmut T. Kandemir and Chita R. Das
*The Pennsylvania State University*
*University Park, PA 16802*
{*jbk5155, arjomand, drg217, kandemir, das*}*@cse.psu.edu*

***Abstract*—Although resistive RAM (ReRAM) technology offers a good combination of high capacity and low-power for cache memories, its long write latency and low endurance are potential showstoppers to its wide commercial adoption. In particular, its low write-endurance can cause fast wear-out of cache lines, bringing reliability issues and leading to capacity reduction over time. This problem is exacerbated when ReRAM cache has dynamic NUCA structure, where each core brings most of its data to the cache banks close to itself and writes become localized. We propose Re-NUCA, a NUCA architecture design for ReRAM cache to address its lifetime problem while keeping its performance high. Re-NUCA relies on performance-wise data criticality: if it realizes a cache line is performance critical, it keeps it in the banks close to the target core, like dynamic NUCA; otherwise, it maps cache lines onto banks using static NUCA to evenly distribute writes over cache banks. This change in mapping of cache lines to banks relaxes the lifetime problem in ReRAM NUCA significantly and wear-levels the lifetime of banks. Re-NUCA needs a logic for detecting performance-wise critical cache lines and a low-overhead changes in TLB for keeping mapping information. Our experimental results of a 16-core chip multiprocessor with 32MB ReRAM L3 cache show that Re-NUCA improves the lifetime of the non-volatile cache by about 42%, on average, with almost no impact on performance.**

***Keywords*-Resistive RAM (R-RAM) based caches, NUCA, wear-leveling, wear-out, S-NUCA, R-NUCA.**

## I. Introduction

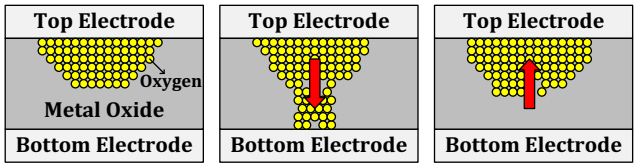
Workloads in the next generation of large-scale computing systems are expected to be highly data-intensive and have large working-sets. The processing power is also steadily increasing and major manufacturers are planning to integrate hundreds of cores on a die. To mitigate performance loss due to increasing memory access rate in multi-core systems running multiple workloads, computer architects tend to employ high-capacity on-chip cache hierarchies. Nevertheless, performance is not the only efficiency metric; an important concern in multi-core systems is total dissipated power. It is known that large last-level cache (LLC) is a major source of on-chip power consumption in chip multiprocessors (CMPs) because they occupy a large portion of processor die and standby power is up to 80% of their total power [5]. Recently, researchers have extensively studied the use of non-volatile memory technologies in large cache designs [14], [11], [8], [10], [15], in contrast to charge-based technologies (SRAM or DRAM), non-volatile memories have near-zero standby power. Among the available non-volatile technologies, resistive RAM (ReRAM) has attractive features as a replacement of SRAM in caches. Specifically, ReRAM has fast read access latency, gives about four times higher density than SRAM and is fully compatible with core fabrication process. These features make it suitable to be employed as baseline technology for LLC in deep cache hierarchies.

Compared to SRAM, write operations in ReRAM are slower and consume more power and prior work [12], [18] has mainly concentrated on alleviating the write performance and write energy issues with ReRAM. However, little attention has been paid to the problem of limited write endurance in ReRAM caches. Indeed, even though ReRAM has typically higher cell endurance (about $10^9$ writes [17]) compared to competitive technologies like STT-RAM, it is still low for cache memories when write traffic of the application is high. This paper studies the lifetime problem in high-capacity ReRAM caches and proposes a low-overhead and reasonable architecture to relax it.

Large caches in modern multicore processors are usually structured as *non-uniform cache architecture (NUCA)*. NUCA is a multi-bank cache where each bank is connected to one core (the number of banks is usually kept equal to the number of cores) and a switched network handles data movement between banks. NUCA caches are organized as either *static NUCA (S-NUCA)* or *dynamic NUCA (D-NUCA)*. In S-NUCA, a cache block(line) is mapped to the cache banks using a subset of bits in address and hence bank assignment is fixed. In D-NUCA, on the other hand, each cache block can be in any bank and the switched network allows data to migrate across different cache banks – that is, if a cache block is frequently used by one core, D-NUCA brings it to the local bank for future fast access. D-NUCA offers lower access latency, but it may *exacerbate the lifetime problem in ReRAM caches* because data migration between banks increases the write traffic into the cache. Moreover, if one program is highly write-intensive, it is highly probable that cache banks close to it get higher write traffic and wear out faster than others. Therefore, ReRAM D-NUCA caches may have lower lifetime than S-NUCA and sometimes lower performance in long execution of the workload.

We propose *Resistive NUCA* architecture (shortly Re-NUCA) that mitigates fast wear-out of cache banks in D-NUCA ReRAM while keeping its performance high. Re-NUCA is designed on top of *Reactive* NUCA (R-NUCA) cache [4], which is a realistic implementation of D-NUCA. R-NUCA allows data migration in NUCA but limits it to few banks close the target core (i.e., those that are only one-

1530-2075/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPS.2016.79




(a) Cell (b) SET (ON state) (c) RESET (OFF state)

Figure 1: Re-RAM cell and its SET and RESET operations.

hop away from the local bank) and reduces the overhead of metadata required for cache block mapping. Our proposed NUCA architecture (Re-NUCA), on the other hand, uses a "hybrid mapping function" based on criticality of the cache block[1]: *it maps the performance-critical data like R-NUCA to keep the data close to the cores, and spreads-out non-critical cache blocks to other banks (like S-NUCA).* In this way, Re-NUCA tries to reduce write intensity on cache banks by spreading them over the entire cache space, while it offers low latency by keeping the critical cache lines in the banks near to the core. Having this hybrid mapping function, Re-NUCA thus relaxes write intensity onto cache banks (almost the same as S-NUCA) while keeping performance high (close to that of R-NUCA).

Implementing Re-NUCA requires a mechanism to capture the criticality of the cache blocks. Re-NUCA determines the criticality of each cache line at the instruction level using a simple *criticality predictor* which works on the heuristics of the instruction issuing a data fetch. In addition, this architecture needs a hardware to choose proper mapping function when searching or allocating a cache line (either S-NUCA or R-NUCA mapping). Re-NUCA achieves this goal by adding few metadata bits to TLB, so (1) it reduces the overhead of this structure by avoiding to store address tag of the cache blocks, and (2) the controller of the ReRAM cache knows which function has to be used prior to access to the cache (since TLB search is performed in early cycles of memory access and the mapping information is available when accessing LLC).

This paper makes the following main **contributions**:

- We propose Re-NUCA, a novel D-NUCA implementation customized for Re-RAM cache memories. It uses a hybrid of R-NUCA and S-NUCA mapping schemes, with the goal of wear-leveling the last-level caches in a performance-conscious manner. Specifically, R-NUCA is used for *critical* cache blocks, and S-NUCA is used for *non-critical* cache blocks.
- To capture the criticality of a given cache line, we use a criticality predictor that determines how much a cache line is critical to the performance of the processor based on the heuristics of the instruction issuing a cache line fetch. We also suggest to keep the metadata information related to mapping function in TLB, to reduce the overhead of mapping tables and remove its access time from critical-path latency of the processor.
- We evaluate the performance and lifetime of Re-NUCA using a large set of multi-programmed workloads with different levels of memory/write intensities. Our experiments show that Re-NUCA improves the lifetime by 42%, on average, without loosing performance over R-NUCA.

[1]A cache block is assumed to be *critical*, if it contains one word (or more words) that the core needs them in short time to avoid pipeline stall.

## II. Background

### A. Resistive RAM

Resistive memories, in general, refer to any technology that uses a variable resistance to store information. However, ReRAM in this paper refers to a subset of memories that use metal oxides as the storage medium, also called as metal-oxide ReRAM.

As shown in Figure 1a, a ReRAM cell consists of a metal-oxide layer sandwiched between two metal electrodes, named top electrode and bottom electrode. The cell can be either in low resistance state (i.e., SET or "1") or high resistance state (i.e., RESET or "0"). In order to switch the state of a ReRAM cell, an external positive voltage with specific polarity, magnitude and duration has to be applied to the sandwiched layer through the electrodes. The SET and RESET operations are shown in Figure 1b and Figure 1c, respectively. When a positive biased voltage is applied to the top electrode, the metal ions (or oxygens) are forced to migrate through oxide, and eventually reach the bottom electrode. The ion-path is highly conductive, and the cell's equivalent resistance value is low (SET). The low-resistance state changes again to a high-resistance state by positively biasing the bottom electrode (RESET).

The biggest advantage of the ReRAM is its good compatibility with the CMOS process used in fabrication of logic (cores). Furthermore, the voltage required for the ion-path formation has a linear relationship with the oxide layer thickness – that is, the required voltage will decrease with a decrease in the thickness. This makes ReRAM a highly-scalable and promising alternative to SRAM. The challenging issues with ReRAM are high write latency, high write energy and low cell endurance (in terms of number of writes). A few prior studies target performance and power issues related to write operations on ReRAM when used as cache and main memory [12], [18]. This paper focuses on the limited write endurance of ReRAM. Current prototypes show that a ReRAM cell can have an endurance of $10^9$ [17] to $10^{11}$ [6], [7], [1] writes. Although this per-cell endurance is large, it can be a source of failure in cache memories when running applications are write-intensive.

### B. NUCA Architecture for Large Caches

Because of small size of ReRAM cell, the on-chip ReRAM caches usually have large capacity and are subject to optimization techniques used for large caches. Large caches are usually structured as NUCA where the entire

cache is partitioned into multiple banks. Each bank is connected to one core and an on-chip network for data and address transfer between banks. NUCA exhibits varying access latencies depending on the distance between the data and the core requesting it. In static NUCA (S-NUCA), mapping a cache block to banks is fixed and is determined using the lower bits of the block's address. This makes redirection (finding the target bank on requesting a cache block) in S-NUCA simple and obviates the need for any lookup table. In D-NUCA, any given line can be mapped into several banks, and a cache block can migrate between banks according to the access frequency – that is, frequently used cache lines migrate to banks closer to the core. D-NUCA needs a table to keep the redirection data for each cache block that is kept along with coherency information in the directory. On a cache access, the directory is checked for coherency issues, and if it is a hit, the associated redirection information is also read to determine the bank index currently holding the cache block.

Accessing D-NUCA for this metadata information increases the energy consumption of the directory and increases the traffic of the switched network. This clearly complicates the implementation of D-NUCA. Considering the overheads of migration to keep the data close to the core requesting the data like in D-NUCA, Hardavellas, et al. proposed *Reactive* NUCA (R-NUCA), [4] which tries to combine the benefits of both S-NUCA and D-NUCA. In R-NUCA, cache blocks for each core are allowed to be stored in a fixed-size cluster that includes banks that are (at most) one hop away from the core. Figure 4(a) pictorially shows bank-level clustering in R-NUCA for a cache with 16 banks and 16 cores. In this example, the cache blocks requested by core are allocated in the shaded region. As can be observed, shaded region cache banks are at most one hop away from the requested core. Thus cache lines accessed by each core are always close to it (at most one hop away from the target core), which resembles D-NUCA in performance. Similar to S-NUCA, the address redirection in each cluster is done by simply decoding few low-order bits for the bank index. The mapping function used by R-NUCA is:

$$DestinationBank = (Addr + \overline{RID} + 1)\&(n-1),$$

where $RID$ is the rotational ID in [4] and $n$ is the cluster size which in this case is 4.

## III. Motivation

If writes were uniformly distributed over available cache banks, the cache banks would be wearing out at almost the same rate, and the per-cell endurance limit of ReRAM would result in a long lifetime for the cache. However, write accesses as well as memory accesses can exhibit significant *non-uniformity* in real workloads, especially when programs running on different cores are different and have different memory intensity levels. Thus, in D-NUCA or its variants like R-NUCA, the cache lifetime is substantially degraded because cache banks closer to the cores running memory-intensive programs wear out fast than those closer to cores running memory non-intensive (or less-intensive) programs.

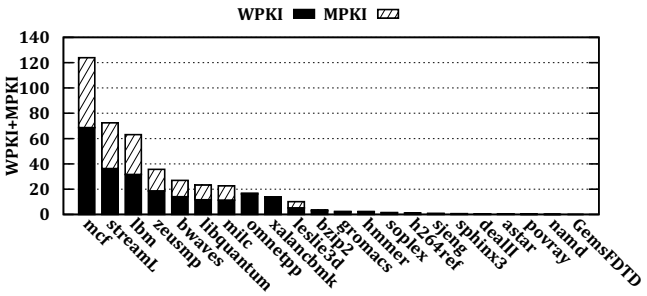



Figure 2: WPKI and MPKI for the studied applications.

To demonstrate the importance of NUCA architecture and its mapping scheme on the write count distribution over cache banks, we performed a series of lifetime and performance evaluation in a system with the configuration given in Table I. The L3 cache (last-level cache) is made up of ReRAM, has 16 banks each with 2MB size, and has the NUCA structure with 4×4 on-chip network between cache banks. The system has 16 cores and runs a workload of 16 single-threaded application from the SPEC CPU 2006 suite [13]. Our multi-program workloads include applications with diverse write intensities and the number and pattern of writes can vary greatly from core to core, depending on the application. Figure 2 plots the Writebacks Per Kilo Instruction (WPKI) and Misses Per Kilo Instructions (MPKI) for different applications used in our evaluation. As writes to the L3 caches come from both write backs from L2 and a cache line fetch upon a L3 miss, Figure 2 shows the LLC write intensity of various applications when an application runs individually with a 256KB L2 and 2MB L3 caches.

Figure 3 shows the lifetime variation between banks of the L3 cache for different NUCA architectures, over all evaluated workloads. We evaluated *S-NUCA*, *R-NUCA*, *private cache* (each core has a 1MB private L3 cache) and a cache architecture with perfect wear-leveling scheme (named *Naive* and discussed later in this section). The numbers presented in y-axis are the harmonic mean of lifetimes across all the workloads that is calculated as follows: we run 10 workloads of varying memory intensities and calculate the lifetimes experienced by the cache bank over all these workloads. The harmonic mean lifetime of a cache bank is the harmonic mean of these lifetimes. As discussed earlier, S-NUCA evenly stripes the memory space across all cache banks in the system, so that every core will access all the banks. We expect writes being more uniformly distributed over cache banks in S-NUCA, when compared to two other designs (R-NUCA and Private). The results plotted in Figure 3 confirm this finding, as all cache banks have very similar lifetime in the S-NUCA architecture, regardless of the memory-intensity of the workload bound to each core.

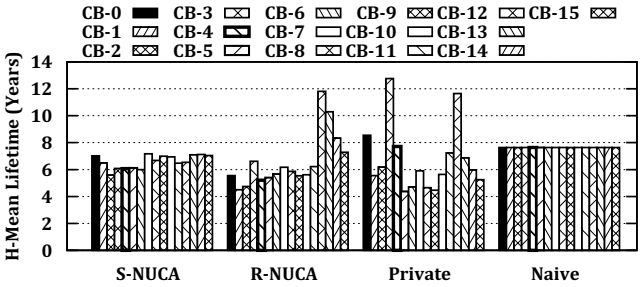
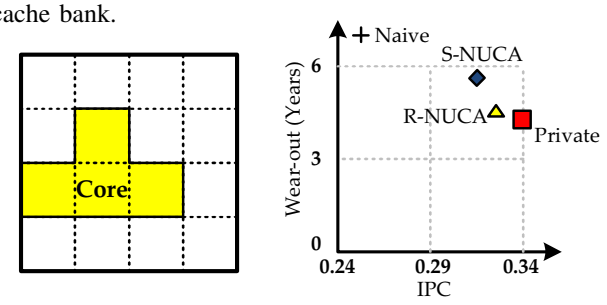



Figure 3: Harmonic mean lifetime in years. CB refers to a cache bank.




(a) (b)

Figure 4: (a) The shaded region is R-NUCA region for a given core, (b) Comparison of performance and wearout characteristics of different cache architectures.

The other extreme design is private cache for each core that offers maximum variation in lifetime of the cache banks – that is, the most heavily written cache bank has a lifetime of less than 2 years in our experiments. We also observe that R-NUCA has relatively large variation between lifetime of the cache banks. The reason is that, in R-NUCA, since data blocks of a core are concentrated in a cluster of four banks (compared to a single LLC in the case of private caches), the clusters of banks used by the memory-intensive applications will still wear out more quickly than the clusters used by the low memory- and write-intensive applications.

### A. Perfect (Naive) wear-leveling approach

A performance-agnostic perfect wear-leveling approach would wear-level the cache banks perfectly by ensuring that each cache bank would receive the same number of writes. Such a perfect wear-leveling scheme allows us to compare how well a NUCA scheme performs with respect to cache bank wear-leveling. This scheme needs oracle knowledge about the number of writebacks and misses incurred for every cache bank. Apart from the oracle knowledge about the individual cache bank, this scheme would also require a directory to know which cache bank contains a particular cache line for a cache line look up after a miss in the L2 private cache. The directory overhead for a high capacity last-level cache is significant and hence this scheme is not a feasible option in a commerical processor and, in this paper, we use it just for comparison. We interchangeably use "Naive" to refer to this perfect wear-leveling scheme as it naively accounts only for the lifetime of cache and ignores the performance.

In our implementation, we keep track of the total number of LLC misses and writebacks (i.e., total writes to the cache) for each bank. When a new cache line needs to be written into the cache, the cache controller chooses the bank with the smallest number of writes so far. This approach leads to near-ideal wear-leveling as shown in Figure 3, with 0% variation in lifetimes between banks. However, as this scheme does not consider performance while wear-leveling, it degrades the application performance by 21%, on average, compared to S-NUCA.

### B. Performance versus lifetime of various NUCA schemes

Figure 4 shows the trade-off between performance and lifetime for various cache architectures: *S-NUCA*, *R-NUCA*, *Private* and *Naive*. The numbers presented in Figure 4 are the harmonic mean values of 10 workloads, which are explained in detail in Section V. In Figure 4, y-axis represents the lifetime in years which signifies the number of years beyond which we loose the whole cache capacity, and therefore, a higher number on that axis is better. X-axis represents the Instructions Committed Per Cycle (IPC). A higher IPC value means a higher performance; hence, a larger number on X-axis is better. As can be observed from this figure, the Naive mapping policy, which balances the number of writes and misses achieves a maximum lifetime of more than 6 years, does not fare well in terms of performance. Note also such a perfect mapping policy is not practical as we explained above. While the Naive mapping scheme performs best for lifetimes, private cache banks perform best for performance. However, private caches perform worst in terms of lifetimes as the writes are distributed unevenly across the cache banks. The next best scheme, as can be observed in Figure 4(b), for better wear-leveling is S-NUCA. Since S-NUCA uses address bits to interleave cache lines across various cache banks, it is not a better scheme for performance as it incurs on-chip traffic for accessing cache blocks that are interleaved across the cache banks. Reactive(R)-NUCA proposed in [4] perrforms close to private caches in terms of performance and fares slightly better than private caches in terms of lifetimes. We remark that Even though R-NUCA is good in performance, with time, cache banks wear out and we loose cache capacity without wear-leveling in place thereby hurting the performance.

From above, there is a clear necessity for a new NUCA architecture which fares well in terms of *both* performance and wear-out. In the next section, we propose a NUCA architecture and related policies to achieve this goal.

## IV. RE-NUCA ARCHITECTURE

Ideally, a wear-level and performance-aware NUCA cache should place all the important cache lines in the nearby

cache banks, and spread out all the non-important cache lines across various cache banks. Important data cache blocks are often referred to in the architectural community as *critical cache blocks* and loads that fetch such critical cache blocks are referred to as critical loads. Our proposed architecture, Resistive NUCA or Re-NUCA, allocates the critical cache blocks closer to the core running the application in a region called the Re-NUCA region, while spreading out the cache blocks that are not critical to the performance using S-NUCA mapping. By spreading out non-critical cache blocks using S-NUCA, write-backs to such non-critical cache blocks can also be distributed across cache banks.

When a cache line is brought to the cache for the first time, we assume a cache-line is not critical, hence a cache line is mapped using S-NUCA. This presumption helps us in prioritizing lifetime over performance for a cache line. Later, based on the output of the criticality predictor logic, the decides on the mapping policy used for cache line allocation. In the following we describe data criticality and the logic we used for criticality prediction.

### A. Critical data and criticality predictor

To explain our notion of criticality better, it is important to consider the micro-architecture of current processors. Most of the commercial processors available in the market currently perform out-of-order execution to achieve maximum performance. Even though instructions are executed out-of-order in the processor, they are committed in-order. All out-of-order processors typically employ a special hardware structure called ReOrder Buffer (ROB), shown in Figure 6(a). The ROB contains all the instructions that are being executed, and the instructions at the head of the ROB are committed upon execution. If the instruction at the head of the ROB is not executed, head of the ROB is stalled until the instruction is executed.

A load issued by a processor is considered critical if it *blocks* the head of the ROB. As explained above, since out-of-order processors follow in-order commit, all the other instructions, which are executed and are ready to be committed, are stalled by the blocking load. As a result, a load which stalls the head of the ROB prevents other ready instructions from being committed thereby decreasing the performance of the application. Such loads that block the head of the ROB are defined as critical loads.

Every instruction executed on a processor contains a Program Counter (PC) which is unique for a specific instruction. Since many data-intensive applications spend considerable amount of time executing loops, each instruction in the program is executed multiple times in different iterations on different data. Consequently, the PC can serve as a valuable attribute to predict various properties of a program phase. Prior works [3] have used this PC to predict the criticality of a load. In this work, we employ a slightly modified version

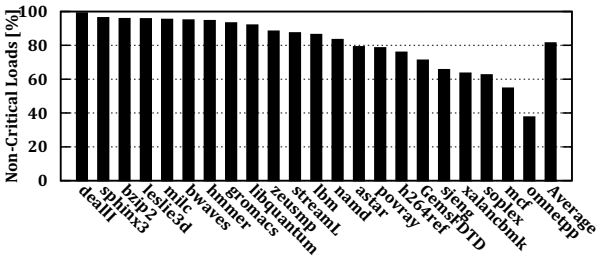



Figure 5: ROB stall percentage.

of the criticality predictor presented in [3], as described later in this section.

Figure 5 shows the percentage of loads that do not stall the head of ROB for various SPEC CPU2006 benchmarks. For these experiments, each benchmark is executed on a 2.4 GHz out-of-order processor containing a private L1(32KB), L2(256KB) and L3(2MB) caches with a DDR-3 memory channel. On an average, over 80% of all loads issued by the processor do not stall the ROB, meaning that non-critical loads contribute to 80% of loads. These results represent a promising direction to identify cache blocks that do not result in performance degradation. That is, a criticality predictor which can accurately predict the non-critical loads is important to identify which cache blocks can be spread over the other banks without incurring any performance loss.

### B. Criticality Predictor

Our criticality predictor adapts hardware structures similar to the Commit Block Predictor presented in [3], which we refer to as the *Criticality Predictor Table* (CPT). More specifically, it contains the following counters:

(1) a PC associated with a load instruction,

(2) a counter similar to robBlockCount in [3] that denotes the *number of times* the ROB has been blocked by the corresponding PC in the past.

(3) a counter numLoadsCount, that indicates the number of loads that were issued by this PC up to this point.

Figure 6(b) shows the operation of the criticality predictor. When a load is issued by the processor, the CPT is indexed with the PC as shown in step 1, and if an entry for the corresponding PC exists, numLoadsCount is incremented, which is referred to in Figure 6 as step 2. If this load results in ROB head block, the robBlockCount counter is incremented which is referred to as step 3. If the CPT does not contain an entry with the corresponding PC, a new entry with the corresponding PC will be inserted into the CPT when the load is committed. The new entry will have numLoadsCount counter set to 1 and a robBlockCount of 1 or 0, depending on whether the load results in a ROB head stall or not.

The difference in our criticality predictor compared to the one proposed in [3] is that we do not need additional information for a PC viz, *LastStallTime*, *MaxStallTime* and *TotalStallTime* as proposed in [3] since we do not have to rank the loads in terms of criticality. Hence, our scheme

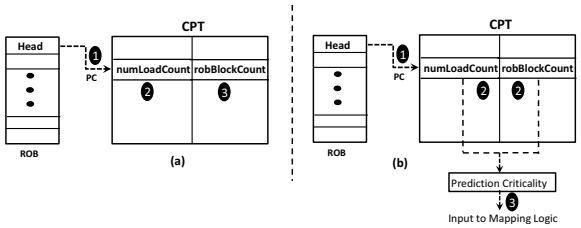



Figure 6: (a) CPT update, and (b) CPT lookup.

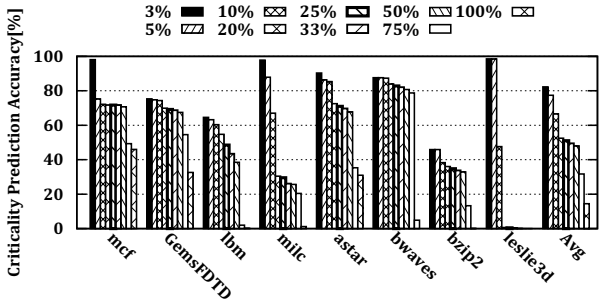



Figure 7: Criticality prediction accuracy.

does not incur any additional overhead as it just needs an extra bit to be sent to the mapping logic to identify if a load being issued is critical or not and averts all the complexity involved in tracking the stall cycles and the corresponding storage.

When a load is issued by the processor, if the CPT lookup results in a hit, we read the robBlockCount and numLoadsCount from the CPT. If the robBlockCount is greater than or equal to a threshold, $x$% of the numLoadsCount, we mark the load as a critical load. This threshold $x$, referred to as the "criticality threshold" in this work, determines the accuracy of the criticality predictor. For example, if the value of $x$ is 100%, then we predict a load as critical if 100% of the load instructions issued by that PC in the past have resulted in a ROB stall. In general, our experiments reveal that a smaller criticality threshold leads to better criticality predictions. A criticality threshold of 100% is a stringent condition. The accuracy of our criticality predictor for different thresholds is plotted in Figure 7. As can be observed, a high criticality threshold of 100% results in a lower accuracy of 14.5%, while a criticality threshold of 3% results in an accuracy as high as 83%, on average with the maximum being around 99% for a workload. Based on this result, we use 3% as our criticality threshold.

Figure 8 shows the percentage of non-critical cache blocks with various criticality threshold values. It shows that, around 50.3% of the cache blocks are fetched from memory are non-critical and do not result in any ROB stall. Figure 9 plots the number of writes to the non-critical cache blocks identified through our criticality predictor. With a criticality threshold of 3%, we observe that around 50% of the writes go to non-critical cache blocks and, as a result, these writes can be distributed across various cache banks to reduce the wear-out without causing the performance to degrade. The

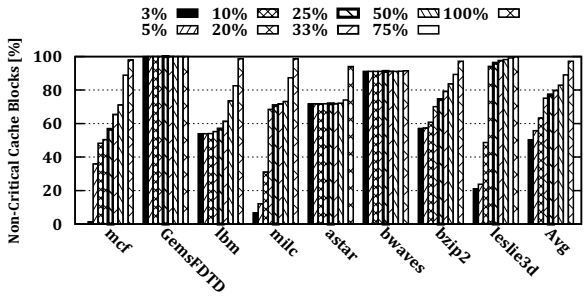



Figure 8: Percentage of noncritical cache blocks.

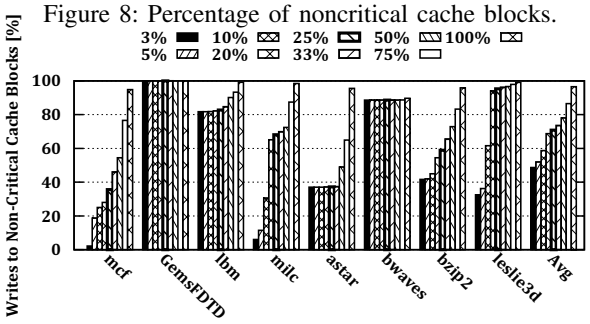



Figure 9: Writes to noncritical blocks.

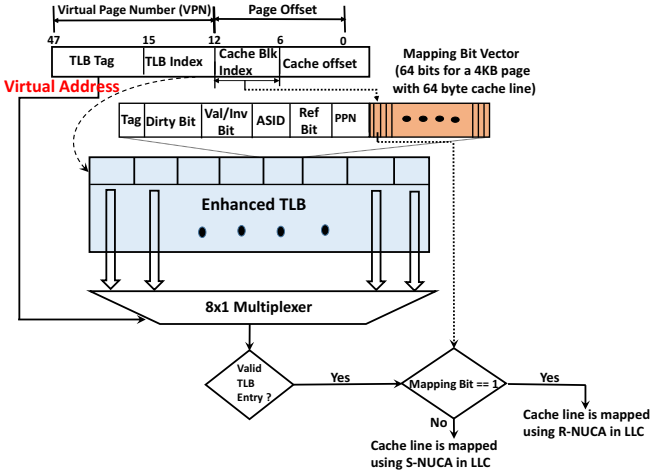



Figure 10: Enhanced TLB with 64 entries, each 8-way showing Mapping Bit Vector for each entry.

non-critical loads shown in Figure 5 are different compared to that in 8 as the non-critical loads in Figure 5 accounts multiple loads to the same cache blocks as critical or non-critical and hence will account to the on-chip cache hits as well. However, the non-critical loads in Figure 8 just accounts for the loads that result in a on-chip cache misses and a memory access, which present an opportunity for cache wear-leveling. Hence, even though Figure 5 indicates 80% non-critical loads on an average, since our mechanism does not consider migration of cache blocks, we can only act on 50.3% of the cache block loads, as shown in Figure 8.

### C. Enhanced TLB

Since a Program Counter (PC) can change from being non-critical to critical and vice versa over the course of

execution, we need to store the mapping information for the cache lines that have already been allocated using one of either S-NUCA or R-NUCA mappings. Upon a miss in the private L1 and L2 caches, the mapping information is used to lookup the corresponding cache bank. We propose an enhanced TLB architecture, which augments a conventional TLB with this mapping information. Every load and store instruction go through the TLB to obtain the physical address of the requested cache line. TLB ideally contains the virtual address-to-physical address translation for a page (typically 4KB), and hence contains information at coarse granularity. Assuming cache line to be 64 bytes, each 4KB page contains 64 cachelines. Since a cache line to be accessed in a page is already part of the virtual address, we can use the same virtual address bits to lookup the TLB entry. In our enhanced TLB architecture, TLB is augmented with a Mapping Bit Vector (MBV) of length 64 bits, each bit corresponding to a cache line in the page. The proposed architecture for Enhanced TLB is shown in Figure 10. The cache line index bits from the virtual address are used as an index into the MBV to update or read the mapping information of a particular cache line.

Upon a last-level cache miss for a cache line, based on the predicted criticality, the corresponding cache line is allocated in cache bank. Once the data is returned to the processor, MBV in the TLB for the corresponding cache line is updated. If the cache line is mapped in the last-level cache bank using S-NUCA (non-critical), then the MBV bit is set to 0; else it is set to 1, indicating that it uses R-NUCA (critical). In our proposal, since a cache line does not change the criticality status in its on-chip lifetime, we do not need to update the MBV bits for a cache line unless the cache line is to be evicted. When a cache line is being evicted, the corresponding MBV bit needs to be reset back to 0.

Our enhanced TLB contains 64 entries in both L1I and L1D per core. Each of them is 8-way set-associative. As each entry in the enhanced TLB contains an extra 64 bits in the MBV, our proposed enhanced TLB architecture adds an extra over-head of 1KB per core, that is 512 bytes for L1I and 512 bytes for L1D TLBs. For a 16 core processor, our enhanced TLB architecture requires an extra 16KB storage, which is less than a single L1I/L1D cache size. Hence, the overhead of our enhanced TLB architecture is negligible in terms both area and power.

We want to emphasize that Re-NUCA tries to achieve the best of both S-NUCA and R-NUCA by combining the performance benefits of R-NUCA by allocating only the critical cache blocks closer to the processor, and wear-leveling of S-NUCA by spreading out the non-critical blocks.

## V. Experimental Evaluation

### A. Evaluation Environment

In this work, we used GEM5 [2] to evaluate our Re-NUCA. We used the system-call emulation (SE) mode of GEM5 instead of the time-consuming full-system (FS) mode. SE mode of simulation in GEM5 is faster compared to the full-system mode. Table I gives our target multicore system. We fast-forward around 2 billion instructions for each benchmark to get to the region of interest, warmup the caches by running 100 million instructions for each benchmark, and then simulate the next 100 million instructions on each core to collect the statistics. We consider ReRAM cache line to wear out beyond $10^{11}$ writes.

| | |
|---|---|
| **Cores** | 16 cores @ 2.4GHz, ALPHA ISA, out-of-order |
| **ROB entries** | 128 |
| **NoC** | 4x4 Mesh |
| **L1I/L1D Cache** | 32KB, 4-way associative, 2-cycle latency, 64 Bytes cache line |
| **L2 Cache** | 256KB (private), 8-way associative, 5-cycle latency, 64 Bytes cache line |
| **L3 Cache** | 2MB per core, 32MB total, 16-way associative, 100 cycle latency, 64 Bytes cache line |
| **Cache Coherence** | MESI Protocol |
| **Memory** | JEDEC-DDR3, 16GB DRAM, 4 channels, 2 ranks per channel, 8 banks per rank, FR-FCFS memory scheduler |

Table I: Simulated architecture configuration.

We used the SPEC CPU 2006 benchmarks with their reference inputs. Table II presents various characteristics of these applications like IPC, last-level cache hit rates, last-level Writes Per Kilo Instruction (WPKI), and last-level cache Misses Per Kilo Instruction (MPKI). As can be observed from this table, these applications exhibit quite a variation in performance; some are memory intensive while others are compute intensive. Based on the sum of the WPKI and MPKI values shown in Table II, we characterize our applications as high, medium and low write intensive. Applications with sum of a WPKI and MPKI greater than 10 are categorized as high write-intensive ; applications with a sum between 1 and 10 are categorized as medium write intensive while all the applications having a sum less than 1 are categorized as low write intensive ones.

We further formed 16-core workloads by randomly choosing applications from the high write-intensive ones along with the medium- and low-intensive ones. Since the cache endurance problems mostly occur when high write-intensive applications are run and the imbalance in wearout occurs when the high write-intensive applications are run with the medium- and low-intensive applications, we choose workloads such that we always run high memory-intensive applications with low/medium write-intensive applications.

### B. Results

We present liftime and performance results for Naive, S-NUCA, R-NUCA, Private and compare them with our proposed Re-NUCA scheme. We report the following results across all the NUCA schemes: harmonic lifetime in years, IPC, and raw minimum lifetime in years. Harmonic lifetime in years represent the harmonic mean of lifetime in years across all the workloads per cache bank. Harmonic mean of lifetimes is a better compared to an average as average

| Application | WPKI | MPKI | Hitrate | IPC | Application | WPKI | MPKI | Hitrate | IPC | Application | WPKI | MPKI | Hitrate | IPC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mcf | 68.67 | 55.29 | 0.20 | 0.07 | omnetpp | 16.22 | 0.61 | 0.96 | 0.78 | h264ref | 1.09 | 0.08 | 0.93 | 2.00 |
| streamL | 36.25 | 36.25 | 0.00 | 0.37 | xalancbmk | 13.17 | 0.76 | 0.94 | 0.89 | sjeng | 0.52 | 0.32 | 0.41 | 1.16 |
| lbm | 31.66 | 31.46 | 0.01 | 0.53 | leslie3d | 5.24 | 4.86 | 0.07 | 1.33 | sphinx3 | 0.30 | 0.30 | 0.06 | 1.96 |
| zeusmp | 18.57 | 17.13 | 0.08 | 0.54 | bzip2 | 2.89 | 0.69 | 0.76 | 1.63 | dealII | 0.33 | 0.12 | 0.65 | 2.27 |
| bwaves | 14.01 | 12.91 | 0.08 | 0.59 | gromacs | 1.85 | 0.61 | 0.67 | 1.61 | astar | 0.24 | 0.12 | 0.54 | 2.08 |
| libquantum | 11.67 | 11.64 | 0.00 | 0.34 | hmmer | 2.20 | 0.13 | 0.94 | 2.61 | povray | 0.18 | 0.04 | 0.79 | 1.57 |
| milc | 11.31 | 11.28 | 0.00 | 0.71 | soplex | 1.27 | 0.25 | 0.80 | 0.94 | namd | 0.04 | 0.05 | 0.21 | 2.34 |
| GemsFDTD | 0.00 | 0.01 | 0.00 | 1.81 | | | | | | | | | | |

Table II: Applications used in the experiments. IPC values shown are for a single core.

lifetime is significantly effected by the extremes. On the contrary the raw minimum lifetime gives the minimum lifetime of any cache bank in all the workloads. This metric helps us to observe how much one can improve the lifetime of a cache bank, which is under write pressure, under different NUCA configurations. Note that higher values of harmonic and raw minimum lifetimes are better. We use a metric called Instructions committed per cycle (IPC) to evaluate the performance of processor for each NUCA scheme. IPC is an accurate metric for multi-programmed workloads and gives the throughput of an out-of-order processor. Higher values of IPC are better.

Figure 12 shows the harmonic mean lifetimes for Re-NUCA mechanism compared to the other mechanisms. X-axis in Figure 12 represents various NUCA schemes, and Y-axis represents the harmonic lifetime in years. Hence, higher the number on Y-axis the better.

The Naive mechanism which does not consider performance and tries to wear-level the cache banks result in the best harmonic lifetime of around 7.5 years. Also, the variation of lifetimes across all the cache banks is 0, and thus Naive mechanism gives the best wear-leveling possible. One can observe from Table III (Actual Results row) that Naive scheme also has the best raw minimum lifetime of 5 years. The Naive scheme degrades the performance on average by 21% compared to S-NUCA. Hence, all the IPC improvements are normalized with respect to S-NUCA as the Naive scheme performs worse even in the sensitivity analysis. This can be attributed to the fact that the Naive mechanism does not consider criticality of cache blocks while choosing the destination of the cache blocks. At the other end of the spectrum is the private cache configuration which has the worst raw minimum lifetime of 2.3 years, as seen in Table III (Actual Results row) for Private, and huge variations in the harmonic lifetimes across cache banks as can be observed in Figure 12.

The private cache configuration, as explained before, localizes writes and misses to the corresponding cache bank without spreading the writes/misses unlike other NUCA schemes. Consequently, a high memory and write intensive application like mcf wears-out its own last-level cache bank faster than other (non-memory and non-write intensive) applications. Also, since the private cache configuration does not warrant on-chip traffic for last-level cache hits, the performance for private cache configuration is the best compared to the other NUCA schemes tested as can be observed in Figure 11. However, the private cache configurations suffer from the capacity utilization problem as the last-level caches are not shared. This is the reason why we see that IPC is lower in some workloads in Figure 11 compared to R-NUCA. However, the private cache configuration does not suffer from cache-interference, a problem with the shared last-level caches. As a result, private caches incur high IPC in most of the workloads compared to other NUCA schemes and on an average achieves around 8% improvement in IPC compared to S-NUCA and around 4% improvement compared to R-NUCA. The improvements are as high as 16% compared to S-NUCA and 14% compared to R-NUCA for certain workloads, as can be observed in Figure 11.

As the Naive and private cache configurations fall at the two ends of the spectrum, the other NUCA schemes S-NUCA and R-NUCA have mediocre IPC and harmonic/raw lifetime compared to the Naive and Private cache configurations. While S-NUCA contains better harmonic/raw minimum lifetime compared to R-NUCA, R-NUCA gives better performance compared to S-NUCA. On an average, R-NUCA beats S-NUCA by 4.7% in IPC, and S-NUCA has a better raw minimum lifetime of 3.36 years while R-NUCA contains a raw minimum lifetime of 2.38 years. The maximum performance difference between R-NUCA and S-NUCA is 12.8% for a certain workload (Figure 11).

Our Re-NUCA scheme, as can be observed in Figures 11, 12 and in Actual Results row for Re-NUCA in Table III, retains the best of both worlds from S-NUCA and R-NUCA in terms of the raw minimum lifetime and performance. By placing the critical blocks closer to the target core in the Re-NUCA region, our Re-NUCA configuration achieves a performance improvement of 5.2% on average, and up to 6.9% for a workload compared to S-NUCA and equalling the performance of R-NUCA on average. On the other hand, by spreading the non-critical cache blocks using S-NUCA approach, *our Re-NUCA scheme wear-levels the cache in a performance-conscious fashion*. The raw minimum lifetime of Re-NUCA scheme is 3.24 years bettering the R-NUCA raw minimum lifetime by 42%, as can be seen in Table III while retaining it's performance. Once can see from Figure 12 that Re-NUCA wear-levels the cache banks in R-NUCA by increasing the harmonic lifetime of cache banks with lower lifetime such as cache banks cb-0, 1, 2, 4, 5, 6, while reducing the harmonic lifetime of other cache

| Application | Naive | S-NUCA | Re-NUCA | R-NUCA | Private |
|---|---|---|---|---|---|
| Actual Results | 4.95 | 3.37 | 3.24 | 2.38 | 2.32 |
| L2-128KB | 7.14 | 3.9 | 3.09 | 2.31 | 2.31 |
| L3-1MB | 3.64 | 1.67 | 1.67 | 1.38 | 1.38 |
| ROB-168 | 7.06 | 3.26 | 3.26 | 2.33 | 2.32 |

Table III: Raw Minimum lifetimes.

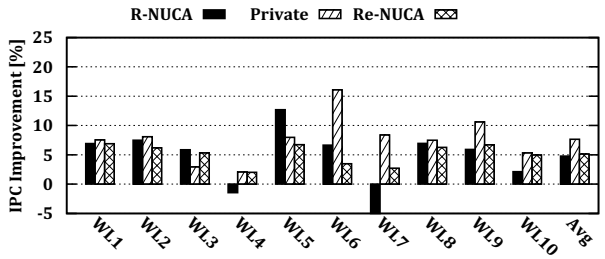



Figure 11: IPC Improvements. All the improvements are normalized to S-NUCA.

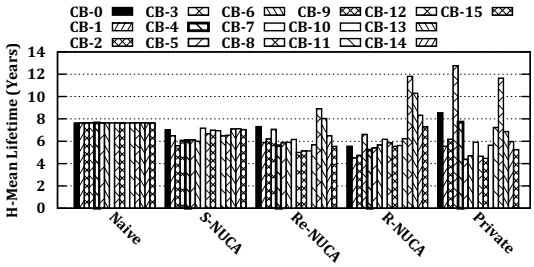



Figure 12: Re-NUCA wearout.

banks which are remarkably high such as cache banks, cb-12, 13,14 and 15. Recall that Re-NUCA employs a hybrid mechanism of R-NUCA and S-NUCA in a performance-conscious manner. If a cache block is predicted as critical, it is allocated in the Re-NUCA region close to the target core, whereas if the cache block is predicted as non-critical, cache blocks are spread-out across last level cache banks using S-NUCA.

### C. Sensitivity Analysis

In this subsection, we discuss how our Re-NUCA works with varying sizes of caches in the cache hierarchy and with the changes in the micro-architecture of the processor especially the number of ROB entries. One of the cache hierarchy parameters which effect the number of writes to the last level cache banks is the sizes of L2 and L3 cache banks itself. Since reducing the size of L2 increases the number of L2 misses and hence the write-backs, we evaluated the impact of Re-NUCA by decreasing the size of L2 to 128KB while our default system had 256KB. Figure 13 and the L2-128KB row in Table III show the harmonic and raw minimum lifetime of Re-NUCA compared to the other schemes. As can be observed, Re-NUCA reduces the variation in harmonic lifetimes across the cache banks compared to R-NUCA, while managing the performance degaradation of only 1.5%, on an average, compared to R-NUCA as can be observed in Table III in L2-128KB row. The raw minimum lifetime of Re-NUCA with 128KB L2

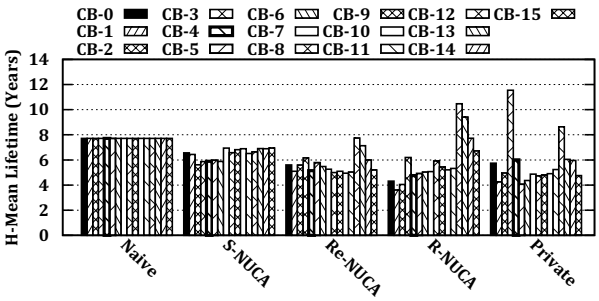



Figure 13: Wear-leveling with an L2 size of 128KB.

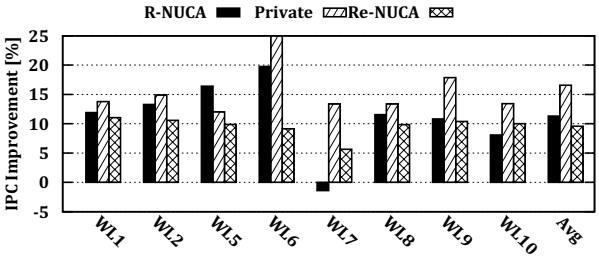



Figure 14: IPC improvements with L2 size of 128KB.

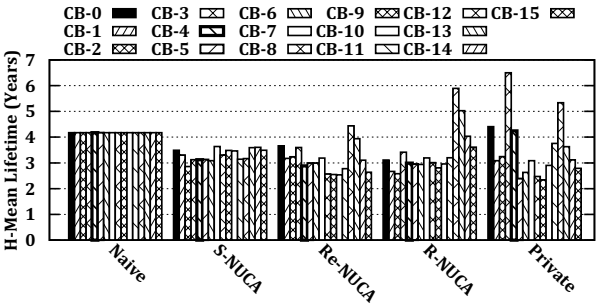



Figure 15: Wear-leveling with an L3 size of 1MB.

is 3.10 years compared to R-NUCA with 2.3 years thereby improving the raw minimum lifetime by 34.8%.

As our next sensitivity experiment, we reduce the size of L3 Re-RAM cache bank to 1MB while our baseline consists of 2MB per cache bank. With the decreased L3 cache bank size, the memory-intensive application incurs more L3 cache misses, which results in fetching more cache blocks and thus increasing the writes to the L3 cache bank. The harmonic mean and lifetime improvements are plotted in Figures 15 and in L3-1MB row in Table III, respectively. As can be observed, Re-NUCA wear-levels the lifetimes similar to L2 with 128KB case. Re-NUCA improves the raw minimum lifetime compared to R-NUCA from 1.38 to 1.67 years improving it by 21%. On average, Re-NUCA improves performance compared to R-NUCA by around 1.8%. It also increases performance by 4.11% compared to S-NUCA on an average. However, the maximum improvements acheived by Re-NUCA compared to R-NUCA and S-NUCA observed are 8.2% and 6.81% respectively.

The other micro-architectural characteristic that influence the impact of Re-NUCA is the number of entries in the ROB itself. With the increased number of entries in the ROB, some of the loads might endup not stalling the

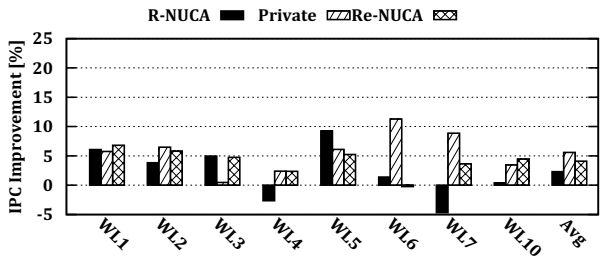



Figure 16: IPC improvements with L3 size of 1MB.

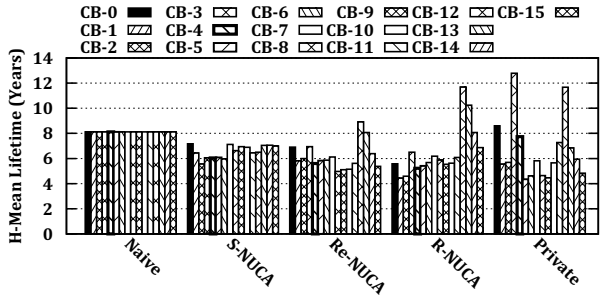



Figure 17: Wear-leveling with an ROB of 168-entry size.

ROB, thereby effecting the criticality predictor. Next, we conducted experiments by increasing the number of entries in ROB to 168, while in our baseline configuration ROB contained 128 entries. With increased ROB size, as can be observed in Figure 18, IPC improves by 5.2% compared to S-NUCA, while it is slightly better than R-NUCA by 0.5%. However, the raw minimum lifetime improves from 2.33 years to 3.26 years compared to R-NUCA, improving its lifetime by around 39.9%, as opposed to 42% with an ROB containing 128 entries.

## VI. Related Work

EqualChance by Mittal and Vetter [9] moves write-intensive cache blocks to a different set in a cache. It keeps track of the writes per set and redirects write to another clean or invalid location if the number of writes goes over a threshold. i2wap (Wang et al.) [16] is a cache management strategy that balances writes between sets and within a set. They combine a main memory wear-leveling technique for addressing variations between sets using a technique to reduce variation within a set.

In our work, we target inter-cache bank wear-leveling while the works mentioned above try to wear-level a cache bank at a finer granularity at inter-set and intra-set level. Though our approach is orthogonal to their approaches, their approaches can be complementarily implemented on top of our proposed approach to reap higher benefits in terms of wear-leveling in a performance-conscious fashion.

## VII. Conclusion

This paper tries to wear-level ReRAM last-level cache banks in a perform-conscious manner. We proposed a novel Re-NUCA mechanism, which is a hybrid of R-NUCA and S-NUCA schemes, to wear-level the caches without degrading performance. We also proposed a low overhead criticality

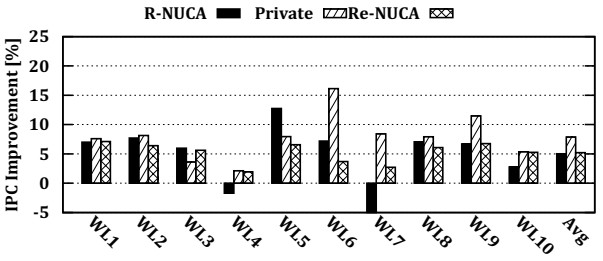



Figure 18: IPC improvements: with an ROB size of 168.

predictor which keeps the performance critical cache lines closer to the processor in the Re-NUCA region using R-NUCA while spreading out all the non-critical cache lines using S-NUCA. Our Re-NUCA scheme improves the minimum lifetime by 42% over R-NUCA and bettering the performance by on an average, 0.5% over R-NUCA, thereby achieving the best of both performance and lifetime.

## Acknowledgment

This work is supported in parts by NSF grants 1526750, 1302557, 1213052 and 1439021, and a grant from Intel.

## References

[1] K. Aratani et al. A novel resistance memory with high scalability and nanosecond switching. In *IEDM*, 2007.
[2] N. Binkert et al. The gem5 simulator. 2011.
[3] S. Ghose et al. Improving memory scheduling via processor-side load criticality information. ISCA, 2013.
[4] N. Hardavellas et al. Reactive nuca: Near-optimal block placement and replication in distributed caches. ISCA, 2009.
[5] C. Kim et al. A forward body-biased-low-leakage SRAM cache: device and architecture considerations. In *ISLPED*, 2003.
[6] Y.-B. Kim et al. Bi-layered RRAM with unlimited endurance and extremely uniform switching. In *VLSIT*, 2011.
[7] M.-J. Lee et al. A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-x/TaO2-x bilayer structures. *Nature Materials*, 2011.
[8] A. K. Mishra et al. Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs. In *ISCA*, 2011.
[9] S. Mittal and J. S. Vetter. Equalchance: Addressing intra-set write variation to increase lifetime of non-volatile caches. INFLOW, 2014.
[10] S. P. Park et al. Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture. In *DAC*, 2012.
[11] M. Rasquinha et al. An energy efficient cache design using spin torque transfer (STT) RAM. In *ISLPED*, 2010.
[12] M. Shevgoor et al. Improving memristor memory with sneak current sharing. In *ICCD*, 2015.
[13] C. D. Spradling. SPEC CPU2006 benchmark tools. *Computer Architecture News*, 35, 2007.
[14] G. Sun et al. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *HPCA*, 2009.
[15] Z. Sun et al. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *MICRO*, 2011.
[16] J. Wang et al. I2wap: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations. HPCA, 2013.
[17] Z. Wei et al. Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism. In *IEDM*, 2008.
[18] C. Xu et al. Overcoming the challenges of crossbar resistive memory architectures. In *HPCA*, pages 476–488, 2015.