MC-ELMM: Multi-Chip Endurance-Limited Memory Management

Andrew M. Bartolo
Stanford University
USA

Mohamed M. Sabry Aly
Nanyang Technological University
Singapore

George Michelogiannakis
Stanford University
Lawrence Berkeley National Laboratory
USA

Subhasish Mitra
Stanford University
USA

ABSTRACT

Non-volatile memories (NVMs) have become a staple of architectural research. NVMs naturally enable techniques such as crash consistency; persistent snapshots, logs, and heaps; security-enhanced memories; and even near- or in-memory processing. However, all current NVM technologies suffer from limited write endurance. Monolithic 3D integration (M3D), an NVM-enabled, near-memory technique, drastically increases compute-to-memory connectivity, improving the energy-delay product (EDP), especially for data-intensive workloads. However, M3D systems have another constraint: scaling M3D memory capacity adds multiple compute-plus-memory chips in a NUMA arrangement. In response, we first develop a lifetime extension mechanism for endurance-limited memories (ELMs) that extends chip lifetime from mere minutes to several years. Our page-based scheme minimizes execution disruption. Second, we extend our single-chip scheme to multiple M3D chips. We show that NUMA policies for DRAM systems are ill-suited for M3D because they either incur too many costly off-chip accesses or sacrifice lifetime. Our technique preserves NUMA locality benefits while significantly improving overall system lifetime. For homogeneous multi-threaded workloads running across multiple NUMA nodes (chips), our multi-chip scheme increases lifetime over our single-chip scheme by a geometric mean of 48%. For a heterogeneous mixture of workloads running on a multi-monolithic-chip cluster, we increase cluster lifetime by a factor of 4.7×, bounded by a 6% energy and 6% runtime overhead (commonly with no runtime overhead at all).

CCS CONCEPTS

• **Hardware** → **Analysis and design of emerging devices and systems**; **Memory and dense storage**; • **Computer systems organization** → *Architectures*.

KEYWORDS

endurance, non-volatile memories, endurance-limited memories, resistive RAM, phase-change memory, multi-chip, multi-node

1 INTRODUCTION

Recently, non-volatile memories (NVMs) have given rise to a host of new architectural techniques. Like flash memory, NVMs retain their contents without power, but add byte-level addressing to enable techniques such as crash consistency; persistent snapshots, logs, and heaps; security-enhanced memories; and even near- and in-memory processing. However, all current NVM technologies suffer from limited write endurance: they are endurance-limited memories ("ELMs").

At the same time, workloads such as machine learning, large-scale simulations, and graph analytics face a "memory wall" [1] in keeping compute units fed with data. We therefore need a new architectural paradigm – one that supports dense connectivity between compute logic and memory. Monolithic 3D integration (M3D) is one auspicious approach for increasing compute-to-memory connectivity. M3D chips integrate multiple layers of memory and compute logic (directly atop each other), drastically increasing connectivity over other forms of integration, including "2.5D" die stacking, bonding, or vertical integration using through-silicon vias (TSVs). As a result, M3D systems deliver significant energy-delay product (EDP) benefits [2–15].

Due to fabrication constraints, M3D systems must use ELMs instead of DRAM. ELMs include phase-change RAM (PCRAM), resistive RAM (RRAM), and spin-torque transfer magnetic RAM (STT-MRAM). Bit cells within these memories can be written only 10^5 – 10^9 times before failure (see Sec. 3.1). A system that naïvely attempts to treat ELMs just like unlimited-write-endurance DRAM will fail within one day's time (see Sec. 4).

M3D systems achieve large energy and execution time benefits by keeping memory accesses on-chip whenever possible. However, in order to scale compute and memory capacity, we must also support multiple M3D chips in a cluster. Many past papers have looked at extending ELM lifetime within a single chip. However, we show that in a multi-chip cluster environment, extending lifetime individually within chips leaves around 80% of cluster lifetime unrealized (see Sec. 8.4). This is because different workloads wear the chips at different rates. Thus, to extend lifetime, we should efficiently remap workloads across chips. Existing work either (i) assumes the presence of off-chip DRAM, which penalizes EDP [8–10], (ii) does not consider multiple chips, or (iii) in a multi-chip system, assumes that all execution must pause while the entire contents of chips' memories are swapped at once. In today's availability- and performance-critical datacenters, pausing execution in such a way is unacceptable.

We are thus tasked with improving lifetime not just on a single ELM chip, but across an entire cluster comprised of ELM chips, with minimal execution disruption. To accomplish this, we modify the virtual memory system to and first-class support for endurance tracking in units of *pages*, instead of entire chip memories. In Secs. 5 and 6.4, we show that our mechanism increases single-chip lifetime by hundreds of times for a 1% energy penalty and 1% runtime penalty, commonly with no runtime penalty at all. Then,

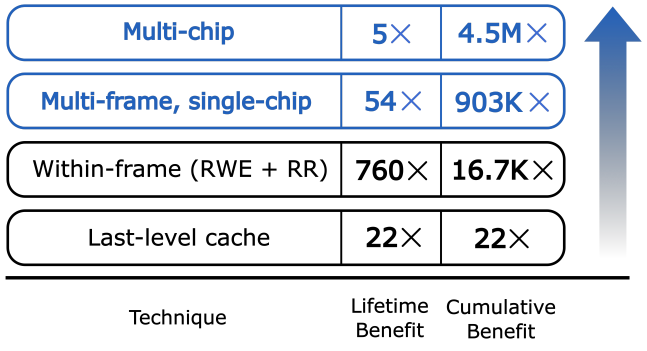


Fig. 1: Multiplicative lifetime benefits from stacking our techniques.

in Sec. 8, we extend our single-chip scheme to a multi-M3D cluster, showing that it extends lifetime a further 4.7× over our single-chip scheme, bounded by an additional 6% energy and 6% runtime penalty (and commonly with 0% runtime penalty).

2 OBJECTIVE

We define lifetime as the minimum amount of runtime required to make *any single bit* in a chip's memory fail. For simplicity, we do not consider complementary, compatible techniques such as backup arrays, error-correcting codes, and ChipKill [16, 17]. We use the term "frame" to refer to the smallest unit of physical memory manageable by the operating system (for example, 4 KiB frames, though we use larger 1 MiB "jumbo" frames to reduce overhead). Similarly, a "page" denotes a unit of virtual memory which maps onto a physical frame. A "line", or "cache line", denotes the smallest unit of memory tracked by a cache. In later sections, in our distributed multi-chip scheme, an individual chip forms a "node".

In Sec. 4, we use a standard LLC to enhance single-chip lifetime and show that meeting lifetime goals with an LLC alone quickly becomes impractical. In Sec. 5, we introduce a new page-level abstraction with bit-level rotation and write elimination. In Sec. 6, we use our new page abstraction and ELMs' byte-level addressability to massively enhance single-chip lifetime without interrupting execution. In Sec. 8, we extend our abstraction further to function across multiple chips, also while minimizing execution interruption. Fig. 1 shows the cumulative effects of these stacked techniques using geometric means across workloads.

3 EXPERIMENTAL SETUP

3.1 Bit Cell Write Endurance

Table 1: Array-level bit cell write endurances.

Ref.	Technology	Year	Array Size	Endurance
[18]	PCRAM	2018	1 Mbit	10^5
[19]	PCRAM	2014	Few cells	10^7
[20]	PCRAM	2011	128 Mbit	10^8
[21]	PCRAM	2013	128 Mbit	10^9
[22]	RRAM	2015	2 Mbit	10^5
[23]	RRAM	2017	16 Mbit	10^6
[24]	RRAM	2019	4 Kbit	10^7
[25]	STT-MRAM	2020	8 Mbit	10^5
[26]	STT-MRAM	2020	32 Mbit	10^6

Table 1 shows array-level bit cell write endurance values for ELMs from literature. Throughout this work, we conservatively assume a bit cell write endurance of 10^6, with which we show multi-year system lifetimes for CPU-based systems (see Sec. 6.4). Achieving our lifetime goals even with a pessimistic 10^6 limit gives us more freedom to select memory technologies for their other favorable properties, such as energy and latency. Device advances beyond 10^6 allow our technique to meet multi-year lifetimes for architectures which generate much more write traffic, such as GPGPUs (see Sec. 7.2).

3.2 Simulated M3D System

Prior work has shown significant EDP benefits for M3D systems [2–15]. Using workloads from Table 3, we performed independent simulations of an M3D system and compared against an off-chip DDR4 baseline, an HBM2 [27] system, and an all-SRAM-main-memory system, and found EDP benefits in line with prior publications. With past work having established EDP benefits, this paper focuses on system lifetime.

Table 2: M3D system specification for lifetime analyses.

Cores	8 cores (MT workloads), 1 core (ST workloads); 2 GHz clock frequency; out-of-order execution; 7nm FinFET; 1.8 nJ/inst.; 80 mW/core leakage	
L1d	32 KiB (private); 8-way; 64B lines; 4 cycle access; 0.72 pJ/bit; 1 mW leakage	
L1i	32 KiB (private); 4-way; 64B lines; 3 cycle access; 0.61 pJ/bit; 1 mW leakage	
L2	256 KiB (private); 8-way; 64B lines; 10 cycle access; 1.8 pJ/bit; 3 mW leakage	
L3	1 MiB/core (shared); 16-way; private; 64B lines; 20 cycle access; 2.4 pJ/bit; 100 mW leakage	
Swap Buffer	128 KiB (shared), 10 cycle access; 1.8 pJ/bit, 2 mW leakage	
Memory: **Monolithic 3D**	7nm monolithic RRAM; 8 channels; 28.4 GiB/s/ch. read; 12.8 GiB/s/ch. write; 49 cycle read; 82 cycle write; 2.3 pJ/bit read; 4.7 pJ/bit write; MD1 queueing model; 100 mW leakage	
Network	400 Gbit/s ethernet; 200 pJ/bit (80 W) per direction	1.6 Tbit/s interposer; 50 pJ/bit (80 W) per direction

The M3D system we use for lifetime analyses is shown in Table 2. We use DESTINY [28, 29] to derive the monolithically-integrated RRAM [30] memory access characteristics, including wire and interconnect. First, we run DESTINY using 28nm monolithic RRAM technology parameters from a major manufacturer. These parameters include bit cell area and aspect ratio; bit cell resistances and capacitances for the low- and high-resistance states; SET and RESET voltages, pulse durations, and energies; access transistor width; and access transistor voltage drop. We then scale the 28nm results to 7nm using contacted gate pitch, supply voltage, gate capacitance, and drive current formulas. DESTINY assumes that all write-verify logic resides off of the critical path.

Throughout the paper, we simulate workloads using the zsim [31] architectural simulator, GCC 10 compiler at optimization level -O3, and Debian 11 operating system. For all workloads, we simulate 50 billion instructions while the workload is running at steady state (i.e., past any initialization phases).

3.3 Workload Selection

Our evaluations include the SPEC CPU 2017 [32] suite, as it contains workloads spanning business, scientific, creative, and technical domains. Some SPEC workloads are single-threaded ("`spec_st`"), and some are multi-threaded ("`spec_mt`").[1] All other workloads are multi-threaded. We also evaluate several machine learning workloads for both inference and training: the ResNet-152 [34] convolutional neural network (CNN), a long short-term memory (LSTM) [35] network, a Transformer attention-mechanism [36] natural language processing (NLP) network, and a generative adversarial network (GAN) [37].[2]

For graph analytics, we run breadth-first search (BFS), connected components (CC), maximal independent subset (MIS), PageRank [40], and graph radii algorithms using the Ligra [41] suite. For these workloads, execution characteristics rely heavily not just on the algorithm being run, but on the dataset being analyzed. For this reason, we evaluate each graph algorithm on two distinct inputs: (i) the real-world LiveJournal dataset [42, 43], and (ii) a randomized R-MAT [44] graph.

We use Linux `perf` to measure last-level cache misses per kilo instruction (LLC MPKI), as well as memory read and write bandwidth. We use GNU `time` to measure maximum resident set size (RSS): the maximum amount of physical memory a process has mapped throughout its runtime. All profiling runs were performed on a dual-socket, 8-cores-per-socket Intel Haswell system with standard hardware prefetchers enabled. To normalize results across our single-threaded and multi-threaded workloads, we express memory traffic in units of bytes per kilo instruction. Our results are in Table 3.

The SPEC workloads span a wide range of memory intensities. The machine learning workloads, while bandwidth-intensive, generally have low MPKI values. This is likely because modern machine learning frameworks use batch processing to enhance dataflow to compute units. The graph analytics workloads are especially memory-intensive, with both high read/write bandwidths, and high LLC MPKI values.

In Sec. 7, we analyze general-purpose GPU (GP-GPU) workloads and in Sec. 8.4 we analyze workloads running across a production HPC cluster. In these, we observe similar trends.

Table 3: Workloads and their characteristics.

	Workload	LLC MPKI	RSS (GiB)	Bytes read per kInst	Bytes written per kInst
1	spec_st.bwaves	0.66	11.14	653.22	95.95
2	spec_st.deepsjeng	0.36	6.72	13.27	28.75
3	spec_st.gcc	2.39	7.47	370.47	10.26
4	spec_st.mcf	9.61	3.87	625.13	88.40
5	spec_st.omnetpp	5.71	0.24	230.53	74.65
6	spec_st.perlbench	0.01	0.20	0.55	0.33
7	spec_st.pop2	0.50	1.45	134.10	75.12
8	spec_st.roms	1.61	10.06	440.31	117.77
9	spec_st.x264	0.20	0.16	21.21	11.78
10	spec_st.xalancbmk	0.21	0.47	22.60	9.85
11	spec_mt.cactubssn	4.04	6.57	187.17	52.68
12	spec_mt.fotonik3d	4.57	9.42	771.36	446.18
13	spec_mt.imagick	0.02	6.91	3.71	2.00
14	spec_mt.lbm	0.50	3.15	1304.26	972.20
15	spec_mt.nab	0.04	0.57	19.35	1.40
16	spec_mt.wrf	0.40	0.19	68.48	46.78
17	spec_mt.xz	0.70	15.04	32.77	29.77
18	ml.gan	0.70	0.40	121.56	68.15
19	ml.lstm_inf	0.86	0.74	56.21	7.99
20	ml.lstm_train	3.14	8.76	431.45	210.84
21	ml.resnet152_inf	1.79	1.12	165.65	35.84
22	ml.resnet152_train	2.29	13.59	194.39	64.86
23	ml.transformer_inf	0.54	3.32	88.22	36.96
24	ml.transformer_train	0.32	10.24	50.05	22.98
25	graph.bfs_lj	7.91	2.74	575.20	101.93
26	graph.bfs_r10m	13.90	4.43	771.30	136.94
27	graph.cc_lj	9.52	2.74	436.49	81.77
28	graph.cc_r10m	24.97	4.43	1023.23	115.63
29	graph.mis_lj	7.35	2.74	364.28	73.79
30	graph.mis_r10m	17.47	4.43	722.32	150.19
31	graph.pagerank_lj	29.13	2.74	1029.43	48.59
32	graph.pagerank_r10m	62.91	4.42	2204.71	102.25
33	graph.radii_lj	12.91	2.74	561.64	75.79
34	graph.radii_r10m	34.28	4.42	1370.11	150.96

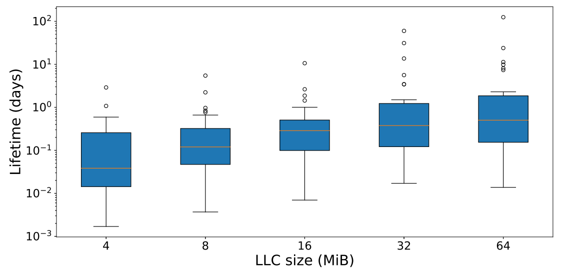


Fig. 2: Single-chip lifetime vs. LLC size (higher is better). Benchmarks from Table 3.

4 EXTENDING LIFETIME BY BUFFERING WRITES

4.1 First Line of Defense: The Last-Level Cache

Write-back caches inherently prevent many writes from reaching main memory. Thus, before considering purpose-built write buffers, we first explore the effect of last-level cache (LLC) size in aiding system lifetime.

Fig. 2 shows overall system lifetime as LLC size is increased. All other specs are as in the M3D system in Table 2, with a 10^6 cell write endurance. The LLC's 16-way set associativity is similar to our real Haswell system's 20-way LLC, and its maximum 8-MiB-per-core size is also similar. Increasing LLC size quickly runs into diminishing returns. Our workloads' RSSes (Table 3) are gigabytes in size; thus, at some point, dirty data must be evicted from the megabytes-large cache. The LLC alone cannot meet a multi-year system lifetime.

4.2 Aside: An Explicit Write Buffer

We now consider the effect of an explicit write buffer [45–47]. Similar to most LLC implementations, we use a single shared write buffer (and not one per core) to coalesce requests across multiple cores. We place the write buffer after the LLC to eliminate writes immediately before they reach memory.

The write buffer allocates lines only upon their eviction from another cache (here, the LLC); a read will not fault a line into the write buffer. Like many LLCs, it uses a physically-indexed, physically-tagged (PIPT) scheme to avoid issues with coherence and aliasing, and is 16-way set-associative.

[1]`spec_mt` contains parallel HPC workloads similar to the PARSEC [33] suite. From the full SPEC suite, we omit the `cam4`, `exchange2`, and `leela` workloads, as their small write traffic volumes obviate lifetime concerns.

[2]We evaluate all machine learning workloads on both the Apache MXNet [38] and Google TensorFlow [39] frameworks, and use whichever implementation produced the higher LLC miss rate.

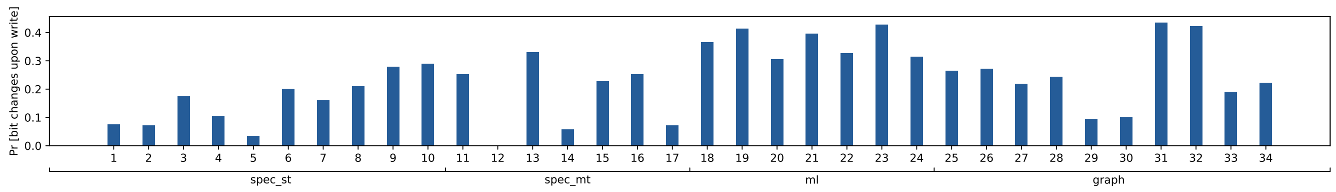


Fig. 3: Bit-toggle probabilities (lower is better).

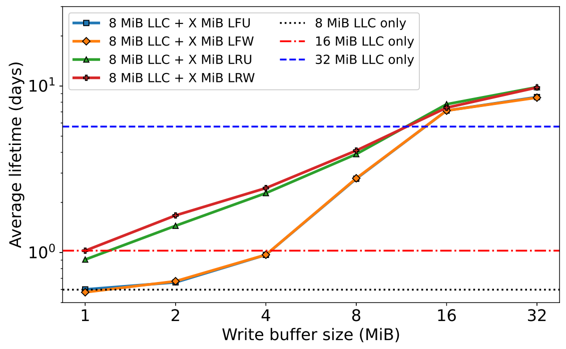


Fig. 4: Single-chip lifetime vs. write buffer size (higher is better).

For the write buffer's eviction policy, we compare LFU (least frequently used, which counts both reads and writes as a "use"), LFW (least frequently written, which counts only writes), LRU (least recently used), and LRW (least recently written). All four of the these policies can be implemented efficiently in hardware: LFU/W, by maintaining a small frequency table within each set, and LRU/W by maintaining a small timestamp table. The simulated M3D system is otherwise configured as in Table 2.

Each write buffer configuration is paired with an 8 MiB LLC. For comparison, we also measure the effect of simply increasing LLC size with no write buffer. Fig. 4 shows our results. The write buffer's effectiveness vs. a similarly-sized LLC drops off substantially at larger sizes. For this reason, we elect **not** to include the write buffer in our simulated system elsewhere in the paper.

5 WEAR-LEVELING WITHIN A MEMORY FRAME

5.1 Prequel: Redundant Write Elimination

Our within-the-frame scheme uses redundant write elimination (RWE) [48] as a subroutine. Before writing a dirty cache line back to main memory, that cache line's old data is read in from memory. Then, circuitry in the memory controller computes a bitmask, where each mask bit is 1 if that bit differs in the cache line and main memory, and 0 otherwise; i.e., `bitmask = (dirty cache line ⊕ line in memory)`. Finally, when writing the cache line back to memory, only bit cells enabled by the bitmask are actually toggled. Table 2's write energy and latency are inclusive of the extra read for RWE.

In Fig. 3, for each workload, we plot the average probability that a bit will be toggled upon a write. Different workloads are shown on the x-axis and are numbered as in Table 3. To collect bit toggle statistics, we use a modified version of DynamoRIO's [49] `drcachesim` [50] tool. In every workload we observe, bit toggle probability was less than 0.5, with a mean of 0.24.

5.2 A Second Observation about Bit Write Patterns

We use RWE to only write bits that "actually" changed in memory. However, some bits may change much more frequently than others; for example, the low-order bits of a counter change exponentially more often than high-order bits. We therefore desire to "spread out" bit writes via rotation, a form of remapping.

[48] rotates the contents of a cache line by randomized byte-aligned values as they are written back to memory. We propose a conceptually-similar subroutine, dubbed "randomized rotation" (RR). Our RR differs from [48] in two ways: (i) We allow shifting by any number of bits, not just 8-bit (byte)-aligned, for greater uniformity, and (ii) our RR metadata is compatible with paged virtual memory, which enables our multi-frame (Sec. 6) and multi-node (Sec. 8) techniques. The combined efficacy of RWE & RR over LLC-only is shown in Fig. 1.

5.3 Hardware & OS Support for RWE & RR

Our goal is to support redundant write elimination (RWE) and randomized rotation (RR) at the page level, to provide an abstraction to base our single-chip and multi-chip schemes on. RWE and RR require minor hardware and software modifications. For RWE, computation and application of the write bitmask can be done with small modifications to the memory controller (Fig. 5), without any intervention from processor cores. Existing ELM macros from major suppliers already contain the bitmask-enable circuitry. For RR, page table entries must be modified to support the addition of the bit-level frame rotation field ("RR value"). The field indicates the bit-level offset by which the frame's contents should be shifted before applying a read or write.

5.3.1 Page Table Entry Size. Fortunately, for RR, the number of bits required to be added to each page table entry is small. In our example system, we use 1 MiB pages. 1 MiB falls in between conventional 4 KiB pages and 2 MiB Linux-x86-64 huge pages [51], which are commonly used to reduce paging overhead. Thus, to represent a bit-level RR value in `[0, frame size in bits)`, we require $\log_2$(1 MiB * 8 bits/byte) = 23 bits.

Consider the page table entry (PTE) structure for x86-64 [52]. Page table entries in x86-64 are word-aligned (i.e., aligned to a

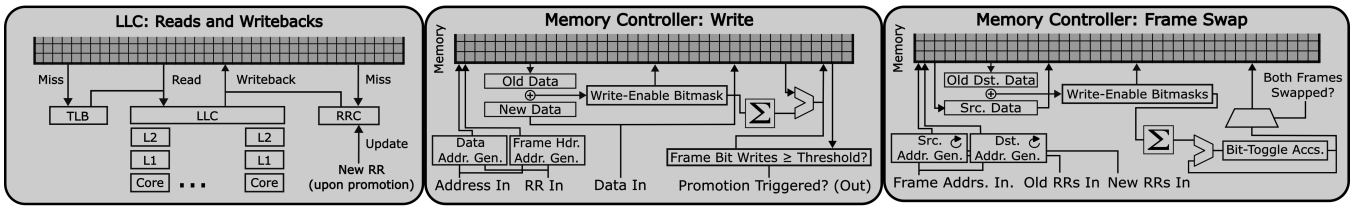


Fig. 5: LLC and memory controller which support RR, RWE, and frame promotion.

multiple of 8 bytes). Thus, many bits in the x86-64 PTE are currently simply ignored or reserved. Specifically, there are 26 such bits. Thus, we can fit our 23-bit RR value into just the ignored/reserved bits, with 3 bits to spare. Our RR metadata therefore does not increase PTE size at all.

5.3.2 Bootstrapping Page Table Lookups. Recall that each PTE now stores the RR value of that page as metadata. However, we cannot look up the RR value of the highest level of the page table ("page directory") from the page containing the page directory itself. To remedy this, for each process, the OS additionally stores the RR value of the page directory. This enables us to bootstrap the virtual-to-physical page table walk.

Of course, not all physical frames are guaranteed to be mapped to a process at any given time. Yet, we still wish to keep track of their RR metadata. For this, the OS additionally maintains a "null page table" of unmapped frames, which acts as a sort of free list. The PTEs within the null page table contain the RR values for all unmapped frames.

5.3.3 Impact on TLB. As in any modern page table implementation, we use a (multi-level) TLB to cache page table entries to speed lookups for virtual-to-physical translations. Each TLB entry must now additionally include the RR bits. For a 1024-entry TLB, using 1 MiB pages, this is an extra (1024 × 23 bits per entry) = 3 KiB.

5.3.4 An RR Cache for LLC Writebacks. Before performing any memory access (read or write), the memory controller must have the RR offset available. For reads, this information is provided via the aforementioned RR field in the page table or TLB. However, writebacks require a different approach. When the PIPT LLC initiates a writeback, the *physical* frame address of the evicted line cannot be used for RR lookups on the *virtually*-addressed page table or TLB.

For this reason, separately from the LLC, we maintain a small RR cache (RRC) that maps physical frame addresses to those frames' RR values. Upon LLC writeback, the RRC is consulted to determine the frame's RR value. Upon an RRC miss, the RR value must be read from memory; specifically, it is read from the frame's *header* region residing at a fixed offset from the start of the frame itself (see Fig. 8). Besides faulting a fresh RR value into the RRC via a read of the frame header, we must also support a simple **update** operation, where the RR value corresponding to a given frame address is updated. Sec. 6.2 details the circumstances of this update.

An RRC miss upon LLC writeback incurs an extra memory read, so the RRC must be highly effective. Fig. 6 shows the weighted mean across benchmarks for RRC hit rate. Even at small sizes and low associativies, the RRC's hit rate approaches 100%. The effectiveness

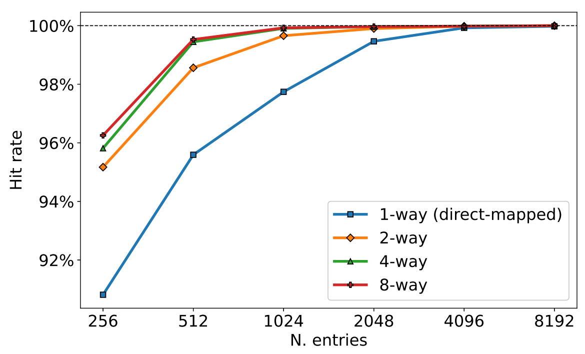


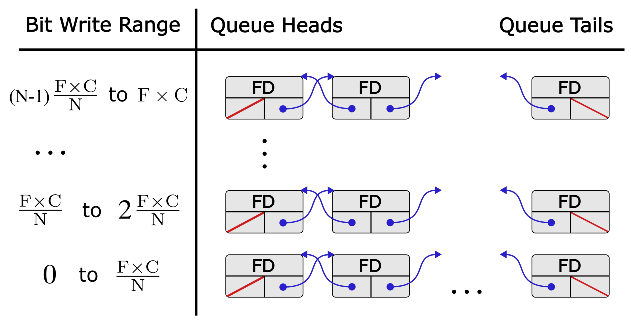
Fig. 6: RRC hit rate sensitivity to size and associativity.

of the RRC can be attributed to the use of jumbo frames; each RRC entry "covers" 1 MiB of main memory. 4 KiB pages require a much larger RRC for an equivalent hit rate; using 4 KiB pages, the RRC sizes required for a 99% hit rate were between 131,072 and 262,144 entries (at any associativity). Using 1 MiB pages, a 4-way, 1024-entry RRC consumes under 10 KiB of SRAM.

6 WEAR-LEVELING AMONG MULTIPLE FRAMES

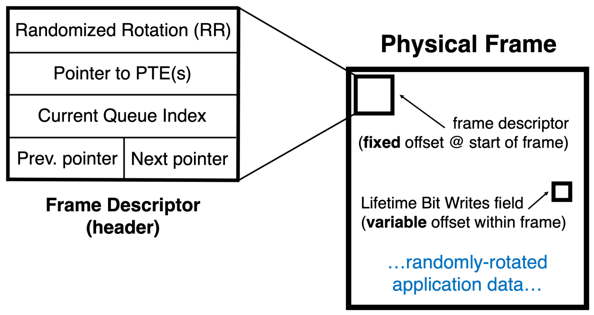
We use RWE and RR to "spread out" bit writes within the frame. Likewise, a similar phenomenon is present at the page level: some pages (e.g., those containing the call stack) will be written more often than others. If we periodically remap these heavily-written virtual pages throughout physical memory, we can increase the lifetime of the system. To that end, we devise a hybrid hardware-software scheme dedicated to managing the wear level of all frames within a chip's main memory. To simplify and reduce its hardware footprint, our scheme leverages standard system main memory for bookkeeping; yet it accelerates some critical-path management operations in hardware to reduce overhead. Crucially, our scheme can extend to multi-chip systems, all while minimally interrupting execution.

Our wear-leveling system operates via a new metadata structure, which resides in the ELM itself. This metadata structure is separate from, and complementary to, our modified page table. In Sec. 6.2, we demonstrate the low endurance overhead of managing metadata directly in ELM.



Bit Write Range
Queue Heads
Queue Tails
(N-1) $\frac{F \times C}{N}$ to $F \times C$
$\frac{F \times C}{N}$ to $2\frac{F \times C}{N}$
0 to $\frac{F \times C}{N}$
FD

Fig. 7: Frame descriptor (FD) queue hierarchy.



Randomized Rotation (RR)
Pointer to PTE(s)
Current Queue Index
Prev. pointer
Next pointer
Frame Descriptor (header)
Physical Frame
frame descriptor (fixed offset @ start of frame)
Lifetime Bit Writes field (variable offset within frame)
…randomly-rotated application data…

Fig. 8: Frame descriptor (FD) data structure within a frame.

6.1 New Metadata Structure

The OS maintains a multi-level hierarchy of queues. Each queue is implemented as a linked list of descriptors of physical frames. The nodes (frame descriptors) within each queue correspond to frames with a similar (within a specified margin) level of write wear. Specifically, "write wear" is the number of bit toggles that have occurred, anywhere in the frame, since manufacture. Fig. 7 shows the queue hierarchy data structure. **F** is the number of bits in a frame, **C** is the cell write endurance, and **N** is the number of levels (queues).

Upon the first power-up of the system after manufacture, all frames initially reside in the first queue. The queues data structure persists across reboots per the use of non-volatile memory. To implement the queues, we use intrusive doubly-linked lists, where each list node is a "frame descriptor" (see Fig. 8). To improve memory reference locality, each frame descriptor is stored as a header within the frame itself, as in Alloy Cache [53] and Unison Cache [54]. Each frame descriptor contains the following:

(1) A randomized rotation (RR) value, indicating the amount by which the frame's non-header contents have been rotated.
(2) A pointer to a PTE, or list of PTEs (for aliased pages), for the virtual page(s) mapped onto the frame. This enables a fast inverse mapping.
(3) The index of the queue that this frame descriptor is currently a member of. This enables the next-highest queue to be found in constant time.
(4) `prev` and `next` pointers for the intrusive linked list.

We also store a "lifetime bit writes" counter for the frame at a variable (randomly-rotated) offset. Unlike other frame metadata fields, this counter changes upon every write, so it is important to spread out its bit writes via RR. Other fields' writes are bounded by **N**, and **N** < **C**, so write wear upon them is not problematic. These fields are therefore stored at fixed offsets within the header.

The objective of the hierarchy of queues is to continuously "bin" frames by their amount of bit write wear. We progressively "promote" frames up the hierarchy as they increase in write wear. When a dirty cache line is evicted from the LLC, the memory controller does the following:

(1) Write the cache line back to main memory, using the RWE subroutine. Though we do require the RR value (via the RR cache) to apply the write, we do *not* shift/remap the entire frame contents at this point; only later, upon frame promotion.
(2) Compute the popcount (Hamming weight) of the bitmask from the RWE. This value equals the number of bits that were actually toggled in memory.
(3) Read the previous lifetime bit writes field from the frame descriptor, and increment it by the new bits-toggled count.
(4) If the frame's new lifetime bit write count exceeds the *threshold value* for its current queue, the memory controller fires an interrupt for the OS to promote the frame.

Threshold values are calculated as follows. First, the system designer chooses a fixed number of queues (levels of the hierarchy), which we call **N**. Then, the threshold value **T** between each level is defined as $\mathbf{T} = \frac{\mathbf{F \times C}}{\mathbf{N}}$.

Each level of the hierarchy represents $\frac{1}{\mathbf{N}}$ of the maximum expected number of bit writes that the frame could sustain with RWE and RR spreading writes uniformly. **N** should be chosen to be high enough to rebalance often for uniformity, yet low enough to not cause excessive write wear from the swaps themselves. In our experiments, we choose **N** = 50,000, as this budgets 5% of overall cell write endurance ($\mathbf{C} = 10^6$) for swap-instigated writes.

6.2 Frame Promotion Algorithm

Upon frame promotion, the OS does the following:

(1) Remove the descriptor for the to-be-promoted frame from its current queue.
(2) Append the promoted descriptor to the tail of the next-higher queue.
(3) Pop the head of the lowest active queue.
(4) Push the descriptor we just popped in (3) onto the tail of the same (lowest) queue.
(5) Swap the memory contents of the frame from (2) with the frame from (4). While performing the swap, shift each frame's contents to its **new** RR value.
(6) Update the "current queue" field of both frame descriptors to reflect their new queues.
(7) Follow the PTE pointer of both frame descriptors to their PTE(s), and update both frames' PTE(s) + TLB entries to reflect their new virtual-to-physical mappings and RR values.
(8) In the RRC, update the RR values for the two frames.

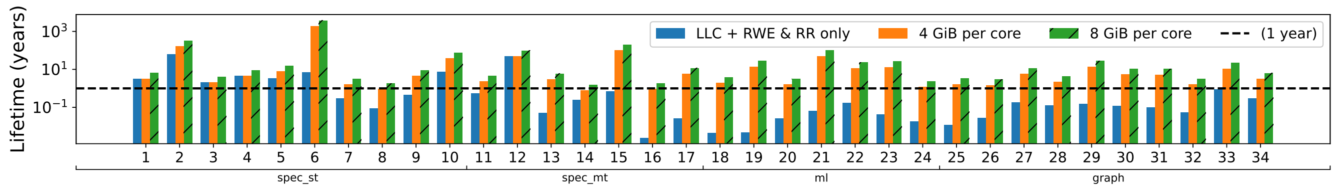


Fig. 9: Single-chip workload lifetimes (higher is better).

Note that we perform steps 3 and 4 (pop and push) to more uniformly select frames for promotion within the same level. This is why each level of our hierarchy is a *queue*.

The promotion process itself does not add significant write wear to the frames. For the most-written frame, promotion only occurs once every $\frac{1}{N}$ fraction of the expected lifetime of the system, and is even lower for all other frames. For example, for an expected system lifetime of 3 years, and $\mathbf{N}$ = 50,000 queues, we promote (swap) the most-written frame once every half an hour. Our simulations pessimistically assume no special write elision when swapping "clean" pages.

Neither do the frames representing the page table experience much additional wear from the updates. `spec_mt.lbm` induces frame promotions more frequently than any other workload, at 135 per second, or once every 7.5 ms. Upon promotion, 11 bytes (8 for virtual-to-physical mapping, and 3 for RR value) must be updated for each of the two PTEs participating in the swap. At 135 promotions per second, this results in an extra 3 KiB/s of write traffic. The data written to update the queues structure is only slightly larger: each of the two frames update their RR (3 bytes), queue index (2 bytes), and 4 × 8-byte pointer fields, for 74 bytes/promotion, which is 10 KiB/s at 135 promotions/s.

6.3 Asymptotics

Step 1 can be done in constant time, as we just wrote the frame, so we have its (fixed-offset) descriptor. Step 2 is also constant-time, as each frame descriptor contains a current queue index. Steps 3 and 4 are constant-time, as we maintain the heads and tails of all queues in an array. The swap in step 5 is constant-time (all frames are the same size). Finally, all frame descriptors contain backpointers to the PTE(s) mapped onto them, with a singly-indirect fast path for non-aliased frames. Thus, page table metadata can be updated in constant time (that of a single pointer dereference plus write) for non-aliased frames, and linear time for aliased frames. The two RRC updates are constant-time.

6.4 Single-Chip Lifetime

We simulate the combined RWE, RR, and frame-promotion algorithms using a system-level in-house simulator written in C++. This simulator takes in zsim execution traces from each workload as input, and outputs frame promotion frequency and lifetime statistics. Lifetimes are shown in Fig. 9. For comparison, the effect of within-the-frame-only techniques is shown in blue. For the combined techniques, we report lifetime as a function of main memory gigabytes per core for several reasons. First, server systems found in datacenters and HPC today commonly have a memory-to-core ratio of around 2 to 16 gigabytes per physical core. We also do so to show that lifetime in our endurance scheme scales practically linearly with the total capacity of main memory. If we have twice as much memory to spread writes out over, we can expect twice the lifetime, and our simulations confirm this.

6.5 Simulated Overhead

To simulate the overhead of our frame-promotion scheme, we make some pessimistic simplifying assumptions. Chief amongst these is that, upon frame promotion, all execution is paused while the promotion process is occurring. In reality, this need not be the case: promotions can be queued and deferred while the main processor runs uninterrupted.

To support frame promotions, we add a small 128 KiB scratch buffer alongside the LLC for swaps (see Table 2). All data that is swapped flows through this buffer only, which eliminates the issue of caches becoming polluted with swap data upon promotion.

We simulate the time and energy required for each promotion by summing the following: (i) the round-trip cost to trap in and out of the kernel to service the promotion-triggered interrupt; (ii) the cost to update all data structures once in the kernel; and (iii) the cost required to rotate and swap the contents of the frames (two frame reads + two frame writes). Using C microbenchmarks, we measure kernel trap round-trip cost on our real Haswell system at 55 µs, and measure kernel data structure update cost at 10 µs. Core energy is calculated per these latencies and energy per instruction; ELM bandwidth & energy (Table 2) determine the cost for the data swap itself. For our most promotion-intensive workload, `spec_mt.lbm`, we observe a 0.9% runtime and 1.0% energy overhead, with lower overheads for all other workloads.

6.6 Promotions Without Interrupting Execution

Frame promotions may be performed in the background. For this, a small interrupt-handling core/unit should be included in the system, so that the main cores can run uninterrupted. Across all workloads, the highest steady-state promotion rate we observe is 135 promotions per second, or once every 7.5 ms. One frame promotion, inclusive of all data transfer and metadata updates, takes around 70 µs. Thus, promotion requests can be serviced in the background, with a bounded queue depth, at steady-state. Though multiple promotions may be triggered within 70 µs of each other, they can be queued to be serviced sequentially, in the background, by the hardware. We perform another experiment measuring the maximum promotion queue depth at any point in the workload's execution under these assumptions. The highest maximum queue depth we observe for any workload is 15, and the mean maximum queue depth is 5. A system designer can thus pick a relatively small fixed size for such a queue, with execution pausing only if the queue becomes full.

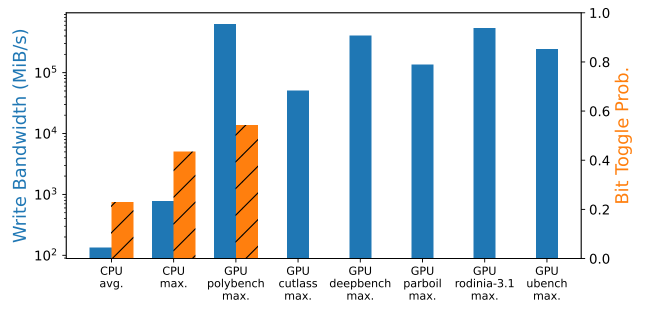


Fig. 10: CPU- and GP-GPU-based write bandwidths and bit toggle probabilities (lower is better).

For these simulations, we use **N** = 50,000 queues, with a bit cell write endurance of **C** = 10^6. The full system specifications are per the M3D system in Table 2. We use a 1-MiB-per-core, 16-way LLC only, with no explicit write buffer.

7 ALTERNATE ARCHITECTURE: GP-GPU

7.1 Workload Characteristics

GP-GPUs, with thousands of simple parallel cores coupled with wide memory buses, make for an interesting point of comparison with CPU-based systems. We measure write bandwidth (in MiB/s), as well as bit toggle probability, as these, along with overall system memory capacity, are the primary determinants of memory lifetime. CPU workloads are from Table 2, with write bandwidth measured on our Intel Haswell system at 2.0 GHz (as in Sec. 3.3), and bit toggle probability simulated as in Sec. 5.1. For the GP-GPU, we use Accel-Sim [55–58] to simulate an NVIDIA Quadro GV100 GP-GPU [59, 60], featuring 5120 CUDA cores running at 1132 MHz and 32 GB of HBM2 memory with 870 GB/s of combined memory bandwidth.

For GP-GPU, we simulate the CUTLASS [61], DeepBench [62], Parboil [63], Rodinia 3.1 [64, 65], and Accel-Sim "µbench" [58] suites, and plot the maximum write bandwidth observed in any kernel within each suite. We were unable to find an easy way to gather bit-toggle statistics for GP-GPU execution. As a remedy, we additionally simulate the Polybench [66] suite, which contains both CPU C and GP-GPU CUDA kernel implementations, and simulate its bit-toggle probability on a CPU (as in Sec. 5.1), and its write bandwidth on the GP-GPU.

Fig. 10 shows our results. The GP-GPU workloads' write bandwidths exceed the largest CPU write bandwidth we observe by around 500×. The absolute-highest GP-GPU write bandwidth we observe, 661 GB/s, represents around 76% of the GV100's 870 GB/s peak combined memory bandwidth. Bit-toggle probability was also higher for the Polybench GP-GPU workload. The GP-GPU's architecture is capable of generating much more write traffic than a typical CPU.

7.2 Lifetime

We now quantify the effect of the GP-GPU's massively higher write bandwidth on system lifetime. We simulate all previous CPU and GP-GPU workloads, using our endurance techniques from

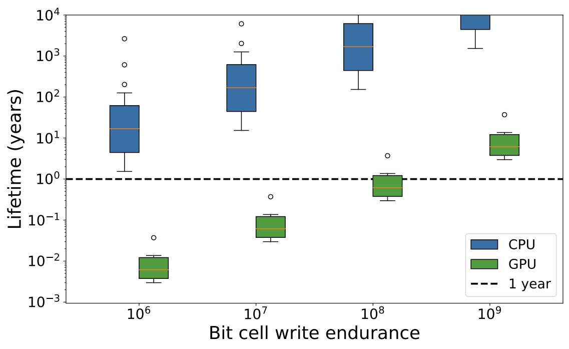


Fig. 11: CPU- and GP-GPU-based lifetimes, using 64 GiB memory (higher is better).

Sec. 5 and 6. The simulated CPU-based system is the M3D one from Table 2. The GP-GPU simulated is the GV100 with endurance-limited memory. To make the comparison more straightforward, the CPU and GP-GPU systems both have the same main memory capacity (64 GiB). For the CPU system, this is equivalent to 8 GiB per core on multithreaded workloads from Sec. 6.4; for the GP-GPU, it is 2× the GV100's memory capacity.

Fig. 11 shows our results. The dashed line indicates a 3-year lifetime. In summary, because GP-GPUs generate massively-higher write traffic, they require bit cell write endurances around 10^9 to achieve multi-year lifetimes, even using our techniques (and are well under one year without them). Because our techniques from Secs. 5 and 6 support general-purpose CPU-facing virtual memory, they can be applied to GP-GPUs as well, with the addition of a small management core (which many GP-GPUs already contain).

8 MULTI-CHIP

8.1 Multi-Node Execution: Defining the Problem

Traditional DRAM systems are commonly arranged in a non-uniform memory access (NUMA) fashion. For any given compute core, some regions of memory can be accessed with lower overhead than others. Typically, this involves different regions of memory being "managed" by different groups of cores ("sockets"), with inter-socket links providing connectivity between sockets. When using multiple M3D chips networked together, we similarly have a NUMA setup.

When running workloads on a NUMA system, we must choose which NUMA pool of memory to allocate pages from. There are two common policies for doing so: 1. First-touch, and 2. Interleave. In the first-touch policy, the page is mapped to the NUMA node of the first core that touches it. This is the default allocation policy of many systems, including the Linux kernel. First-touch is often advantageous because it assumes subsequent accesses to the same page of memory will be primarily made by the same core that first accessed them. If this is indeed the case, then the first-touch policy minimizes off-NUMA-node accesses, which is usually advantageous for execution time and energy.

On the other hand, interleaving seeks to do the opposite: it distributes new page requests across all possible NUMA nodes in

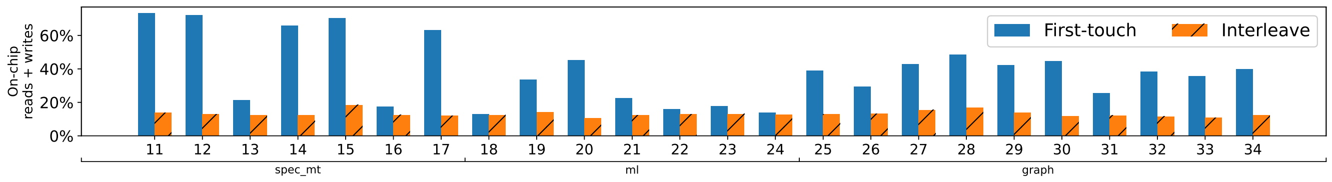


Fig. 12: NUMA on-chip memory access percentage, first-touch vs. interleave (higher is better).

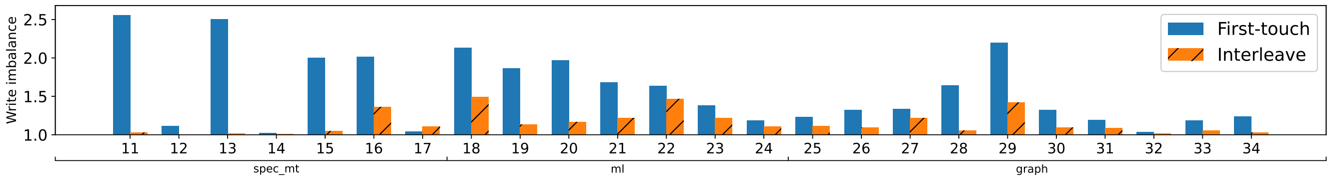


Fig. 13: NUMA write imbalance factor: most-written chip vs. average (lower is better).

the system, usually in a basic round-robin fashion. While this is often bad for locality, resulting in many off-chip accesses, it can sometimes lead to more-even utilization of memory interconnects.

In Fig. 12, we select every multi-threaded workload from our set and simulate them in a NUMA setting. Each workload runs across 8 cores, and each of those 8 cores is assigned to a separate NUMA domain. For each workload, we simulate both a first-touch and an interleave policy. Across the board, we see that first-touch is indeed better at minimizing off-chip accesses. This is particularly crucial in our M3D setting, where off-chip accesses are much costlier than on-chip.

In Fig. 13, we now measure the amount of "write imbalance" incurred by the first-touch and interleave policies. "Write imbalance" is defined as the number of bytes written to the most-written chip, divided by the average number of bytes written across chips. A write imbalance of 1 indicates perfectly-balanced writes.

Here, first-touch loses substantially to interleave. Consider that multi-threaded workloads commonly have one "master" thread, with all other threads being worker threads. Suppose, for example, that the master thread does much more writing than the worker threads. In this case, the chip hosting the master thread will incur higher write wear than the others. This imbalance presents an opportunity for further extending system lifetime.

8.2 Multi-Chip Wear Leveling

Most of the time, to wear-level, we prefer to move pages within the same chip. This lets us avoid the large off-chip bandwidth, latency, and energy penalties. However, as we saw, many multi-node workloads produce an uneven amount of write wear across chips. Therefore, we desire a way to balance the amount of write wear evenly among multiple chips. We combine our insights from Secs. 5, 6, and 6.4 to offer a lightweight scheme for multi-chip wear leveling, featuring the locality benefits of a first-touch policy, but with a lower write imbalance.

Our solution is to adopt a first-touch allocation policy, but to periodically swap the entire contents of one node's memory with another node (while performing process migration [67–69] as well). Because each node has paged memory, the swap may proceed in the background, in units of individual pages, as in VM live migration [70–79]. However, in our later overhead analysis in Sec. 8.5, we pessimistically assume that all execution stops during the swap.

We must now determine how often to swap nodes' memories and which nodes' memories should be swapped. In our single-chip setup, we mapped virtual pages onto physical frames. Here, we likewise perform a mapping of "jobs" onto physical M3D chips. For this, we use a scheme extremely similar to our single-chip one: a hierarchical system of queues (Fig. 7). Instead of frame descriptors, we now have node descriptors; instead of **F**, the number of bits in a frame, we now use **M**, the number of bits in memory for an entire node. We again choose **T** = 50,000 as 5% of our 10^6 cell write endurance **C**; this means a swap will be initiated after every $\frac{1}{50{,}000}$ of the node's overall write endurance.

Each node's memory controllers maintain a counter, which counts the total number of bits written (after RWE) to that node's memory, for its entire lifetime (since manufacture). Whenever a frame is written anywhere on that node, we increment the counter by the number of bits toggled by that frame write. When the counter exceeds its threshold value, the memory controller fires an OS interrupt to promote the node in the queue hierarchy. Similarly to our single-chip scheme, promotion involves a swap of memory contents.

All nodes' OSes maintain a coherent copy of the node descriptor hierarchy. Promotions happen infrequently enough that coherence overhead is not a concern (Sec. 8.5). When a promotion is triggered, the triggered node's OS does the following:

(1) Remove its own node descriptor from its current queue.
(2) Append that descriptor to the tail of the next-higher queue.
(3) Pop the head of the lowest active queue. This chooses the "target" node.
(4) Push the target node descriptor onto the tail of the same (lowest) queue.
(5) Broadcast to all other nodes that the triggered node and target node will swap.
(6) Swap the triggered and target nodes' memory contents.
(7) Update the "current queue" field of both node descriptors to reflect their new queues.

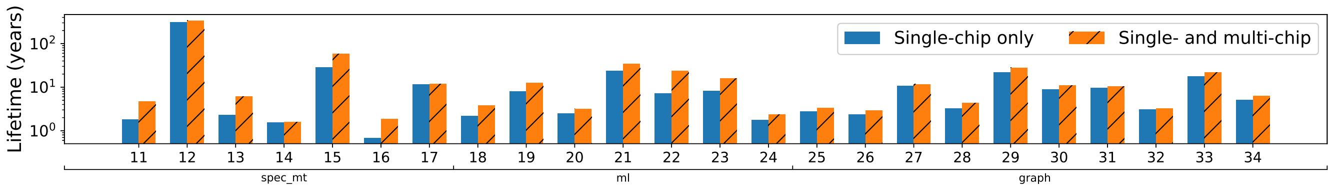


Fig. 14: System lifetimes, with multi-chip wear leveling disabled vs. enabled (higher is better).

(8) Broadcast to all other nodes that the swap is complete.

Each node descriptor contains the following:

(1) The index of the node it represents.
(2) The index of the queue that this node descriptor is currently a member of.
(3) `prev` and `next` pointers for the intrusive linked list.

Because each node maintains its own lifetime bit write counter, this value need not be included in the node descriptors.

Since the number of nodes in a system is generally assumed to be significantly fewer than the number of pages on a single node, each node can maintain a copy of the node descriptor hierarchy in memory with low overhead compared to the single-chip scheme. Alternately, a centralized controller (or consensus-elected leader) and single canonical database can be used to store the hierarchy. In this case, nodes must still message the controller/leader (which is responsible for serializing those requests).

8.3 Multi-Node Lifetime Benefits

Fig. 14 shows the benefits of our multi-node wear-leveling scheme. Here, we select every multi-threaded workload from our set and simulate them in NUMA mode on our simulator from Sec. 6.4. Each workload is configured to run with 8 cores, and each of those 8 cores is assigned to a separate NUMA domain. Each NUMA domain (chip) has 8 GiB of main memory and uses a first-touch allocation policy. We simulate expected lifetime with our multi-node endurance mechanism disabled, and again with it enabled. For homogeneous multi-threaded workloads running across multiple NUMA domains, our multi-chip scheme increases lifetime over our single- chip scheme by a geometric mean of 48%.

8.4 Heterogeneous Mixture of Applications

Finally, we simulate our multi-chip endurance scheme using data from a real HPC cluster. We collected memory read and write statistics for jobs running on NERSC's "Cori" Intel Haswell nodes over the span of ten days [80, 81]. The CDF in Fig. 15 shows the read and write bandwidths across all Cori's Haswell nodes, sampled every one second. These write bandwidth statistics are a key input to our multi-node endurance simulation. We simulate using 8 nodes with write bandwidth values drawn uniformly from the CDF.

The other two inputs to our multi-node simulation (besides write bandwidth) are memory capacity per node, and bit toggle probability (using redundant write elimination as in Sec. 5.1). Each Cori Haswell node has 32 physical cores and 128 GiB RAM. Because measuring bit toggles requires intensive dynamic binary instrumentation and is not supported by built-in architectural performance counters, we were not able to gather bit toggle probabilities on Cori.

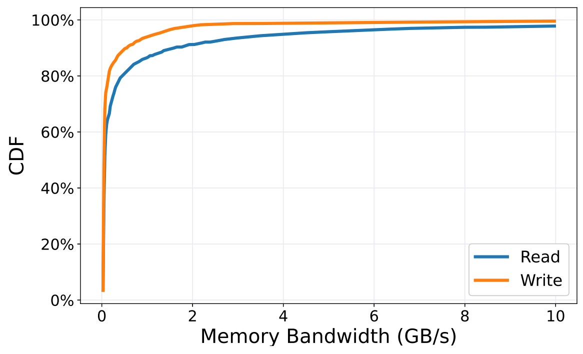


Fig. 15: CDF of node-wide memory read/write bandwidth from NERSC's Cori Haswell nodes over ten days.

In this experiment, we therefore assume a bit toggle probability of 0.5. We justify this conservative assumption by noting that 0.5 is higher than any probability we observe for any workload from Sec. 3.3, and, furthermore, that an average bit toggle probability of 0.5 can be achieved for arbitrary data, by passing the contents to be written through a pseudorandom function (PRF) such as a block or stream cipher before they are physically written.

With multi-node wear leveling disabled, the simulated cluster achieved a **2.5 year** lifetime. With wear leveling enabled, it achieved a **11.7** year lifetime, a 4.7× gain.

8.5 Simulated Overhead

To simplify our overhead analyses, we again pessimistically assume that application execution on all nodes stops during a node promotion. To swap node contents, the swap buffer (Table 2) is again used for temporary space. As in our single-chip scheme, promotion delay is simulated as interrupt, plus data structure update, plus data transfer cost. We use the same 55 µs round-trip interrupt cost, with added 50 µs data structure update cost. For multi-node swaps, off-chip bandwidth (not intra-chip memory bandwidth) is the bottleneck for data transfer. For this reason, we simulate using two different node-to-node interconnects: 400 Gbit/s ethernet at 200 pJ/bit, and a 1.6 Tbit/s silicon interposer at 50 pJ/bit (see Table 2).

Swap energy is modeled by summing several quantities across the two participating nodes. For both nodes, we pessimistically assume that during swap kernel bookkeeping (105 µs trap + data structure update, as above), all cores are spinning at IPC = 1. (In reality, only one core/thread need perform the update.) To calculate data energy, each node (i) reads its entire 128 GiB memory and (ii) transmits that data over the wire, where it is (iii) received into

Table 4: Comparison with related work. Our hardware & algorithms enable *multiplicative* lifetime benefits from the techniques below, with low runtime and energy cost.

	Redundant Write Elimination	Intra- (line, page)	Inter- (line, page)	Physical-virtual remapping	Full paged virtual memory	SRAM/DRAM buffer/cache	Beyond CPU (e.g., systolic, GP-GPU)	Multi-node
[82]	✅							
[83]	✅							
[84]			✅	✅	✅			
[85]	✅							
[86]			✅					
[48]	✅	✅	✅	✅				
[87]	✅		✅					
[88]			✅	✅	✅			
[89]			✅	✅	✅			
[90]			✅	✅	✅	✅		
[91]			✅	✅				
[92]	✅		✅	✅				
[93]			✅	✅				
[94]			✅			✅		
[95]	✅		✅	✅				
[96]			✅	✅				
[97]			✅	✅				
[98]			✅	✅				
[99]				✅		✅		
[100]		✅	✅	✅		✅		
[101]			✅	✅				✅
[102]			✅			✅		
[24]	✅	✅	✅				✅	✅
[103]			✅	✅	✅			
[10]	✅	✅	✅				✅	✅
[104]			✅	✅				
[105]		✅	✅			✅	✅	✅
This Work	✅	✅	✅	✅	✅	✅	✅	✅

the other node's swap buffer and then (iv) written into that node's memory. Thus, each of (i-iv) occurs twice, and all these energies are added to the total. Finally, leakage energies for the duration of the swap (bookkeeping latency plus data transfer latency) are added in.

We simulate multiple independent copies of our most swap-intensive application (`spec_mt.lbm`) running on Cori. The application invokes a node-to-node swap every 71 seconds. For the 400 Gbit/s interconnect, we observe a 5.4% runtime and 5.4% energy overhead, and for the 1.6 Tbit/s interconnect, a 2.7% runtime and 2.8% energy overhead, inclusive of all overheads for our single-chip scheme.

8.6 Promotions Without Interrupting Execution

As with single-chip frame promotions, multi-chip cross-node promotions (memory swaps) can be performed in the background. As in VM live migration, we recommend (but do not require) that each node participating in the swap has around half of its memory free, to enable concurrent buffering of the other node's sent memory contents and any updates. For two nodes with 128 GiB of memory each, connected by 400 Gbit ethernet, the swap delay is around 2.5 seconds. On the cluster, we observe a swap every 71 seconds, so swaps can be performed in the background at steady-state, with a bounded promotion queue depth. In simulation, the promotion queue depth never exceeded the number of nodes (eight).

9 RELATED WORK

Many works have investigated the use of alternative memory technologies as a supplement to, or a substitute for, DRAM. [48, 84, 85, 89, 91, 94, 96–98, 100, 103, 106–109] propose PCRAM or RRAM as either partial or total replacements for DRAM, but do not consider multi-node NUMA systems. [110] analyzes ELM-NUMA systems at the filesystem, rather than the main memory, level. [101] takes an alternate approach to ELM-NUMA lifetime via node bandwidth sharing and mellow writes [99]. [10] and [106] detail wear-leveling schemes that use a very small amount of metadata, but require that every single byte in main memory be shifted periodically. [108] maintains a hierarchy of frames, but does not integrate RWE and RR, and is single-node only.

Flash-based systems also employ wear-leveling schemes [111]. These can be broadly categorized as either static [112–116] or dynamic [117–122]. Static schemes attempt to move cold data to more-worn blocks, whereas dynamic schemes repeatedly reuse blocks with lesser erase counts, but do not attempt to move cold data. Our technique encompasses all frames in memory, and thus incorporates benefits of both static and dynamic wear-leveling. Unlike flash, ELMs such as PCRAM, RRAM, and STT-MRAM do not require high-overhead erase cycles before data can be reused (and consequently do not require garbage-collection routines).

[123] thoroughly analyzes non-ELM NUMA systems and draws the conclusion that, in comparison to then-current wisdom, contention and queueing delays, not wire delays, were responsible for the bulk of NUMA performance degradation. M3D is a drastic-enough paradigm change to warrant revisiting these assumptions, since, in M3D, on-chip communication is vastly more efficient than off-chip. Hardware-assisted page placement [124, 125], Carrefour [123], and AutoNUMA [126] can be thought of as first-touch/interleave hybrids, but both consider DRAM systems only,

and do not attempt to co-optimize NUMA locality with ELM endurance.

Past work on M3D memory endurance has primarily focused on special-purpose machine learning accelerators, and not general-purpose CPU cores [105]. CPU-focused M3D work has thus far not deeply investigated the NUMA locality effect, and how best to preserve lifetime within that constraint. In contrast, our added hardware and software (Sec. 5.3) are structured to *multiplicatively* "stack" (Fig. 1) all techniques from Table 4, providing millions-of-times higher lifetime over naïve execution, while preserving runtime and energy benefits (Sec. 6.5, Sec. 8.5).

10 CONCLUSION

All NVM-enabled techniques, including monolithic 3D integration, must respect the issue of write endurance. M3D significantly improves upon the energy-delay product, but imposes the additional NUMA scaling constraint. Our single-chip scheme delivers hundreds-of-times-higher endurance, achieving multi-year lifetimes even with a small bit cell write endurance of 10^6. Our multi-chip scheme ensures that the performance-favorable first-touch allocation policy does not result in degraded lifetime. In fact, we see the opposite – that the use of multiple chips can enhance overall system lifetime, because they provide more "surface area" over which to wear-level writes. Our combined schemes, with runtime and energy overheads bounded in the single digits, thereby preserve the EDP benefits of monolithic 3D while addressing some of its most challenging limitations.

11 ACKNOWLEDGMENTS

We thank Prof. Heiner Litz and Prof. Sara Achour for their advice and feedback during the early stages of this work. This research was funded in part by the United States Defense Advanced Research Projects Agency (DARPA) 3 Dimensional Monolithic System on a Chip (3DSoC) program, and by the United States Department of Defense (DOD) and Department of Energy (DOE). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the aforementioned funding agencies.

REFERENCES

[1] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM*, 52(4):65–76, apr 2009.
[2] Feihui Li, C. Nicopoulos, T. Richardson, Yuan Xie, V. Narayanan, and M. Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 130–141, https://ieeexplore.ieee.org/document/1635947, 2006. IEEE.
[3] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, Vijaykrishnan Narayanan, Mazin S. Yousif, and Chita R. Das. A Novel Dimensionally-Decomposed Router for on-Chip Communication in 3D Architectures. *SIGARCH Comput. Archit. News*, 35(2):138–149, jun 2007.
[4] Gabriel H. Loh, Yuan Xie, and Bryan Black. Processor Design in 3D Die-Stacking Technologies. *IEEE Micro*, 27(3):31–48, 2007.
[5] Gabriel H. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, page 453–464, USA, 2008. IEEE Computer Society.
[6] Gabriel H. Loh and Yuan Xie. 3D Stacked Microprocessor: Are We There Yet? *IEEE Micro*, 30(3):60–64, 2010.
[7] Yuan Xie. Processor Architecture Design Using 3D Integration Technology. In *2010 23rd International Conference on VLSI Design*, pages 446–451, https://ieeexplore.ieee.org/document/5401205, 2010. IEEE.
[8] Mohamed M. Sabry Aly, Mingyu Gao, Gage Hills, Chi-Shuen Lee, Greg Pitner, Max M. Shulaker, Tony F. Wu, Mehdi Asheghi, Jeff Bokor, Franz Franchetti, Kenneth E. Goodson, Christos Kozyrakis, Igor Markov, Kunle Olukotun, Larry Pileggi, Eric Pop, Jan Rabaey, Christopher Ré, H.-S. Philip Wong, and Subhasish Mitra. Energy-Efficient Abundant-Data Computing: The N3XT 1,000x. *Computer*, 48(12):24–33, 2015.
[9] William Hwang, Mohamed M. Sabry Aly, Yash H. Malviya, Mingyu Gao, Tony F. Wu, Christos Kozyrakis, H.-S. Philip Wong, and Subhasish Mitra. 3D Nanosystems Enable Embedded Abundant-Data Computing: Special Session Paper. In *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*, CODES '17, New York, NY, USA, 2017. Association for Computing Machinery.
[10] Mohamed M. Sabry Aly, Tony F. Wu, Andrew Bartolo, Yash H. Malviya, William Hwang, Gage Hills, Igor Markov, Mary Wootters, Max M. Shulaker, H.-S. Philip Wong, and Subhasish Mitra. The N3XT Approach to Energy-Efficient Abundant-Data Computing. *Proceedings of the IEEE*, 107(1):19–48, January 2019.
[11] Itir Akgun, Dylan Stow, and Yuan Xie. Network-on-Chip Design Guidelines for Monolithic 3-D Integration. *IEEE Micro*, 39(6):46–53, 2019.
[12] Bhargava Gopireddy and Josep Torrellas. Designing Vertical Processors in Monolithic 3D. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 643–656, https://ieeexplore.ieee.org/document/8980321, 2019. IEEE.
[13] Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Donald Yeung, and Bruce Jacob. Design for ReRAM-Based Main-Memory Architectures. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, page 342–350, New York, NY, USA, 2019. Association for Computing Machinery.
[14] Dylan Stow, Itir Akgun, Wenqin Huangfu, Yuan Xie, Xueqi Li, and Gabriel H. Loh. Invited: Efficient System Architecture in the Era of Monolithic 3D: Dynamic Inter-Tier Interconnect and Processing-in-Memory. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4, https://ieeexplore.ieee.org/document/8807061, 2019. IEEE.
[15] Candace Walden, Devesh Singh, Meenatchi Jagasivamani, Shang Li, Luyi Kang, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. Monolithically Integrating Non-Volatile Main Memory over the Last-Level Cache. *ACM Trans. Archit. Code Optim.*, 18(4), jul 2021.
[16] Timothy J Dell. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics division*, 11(1-23):5–7, 1997.
[17] Da Zhang, Vilas Sridharan, and Xun Jian. Exploring and Optimizing Chipkill-Correct for Persistent Memory Based on High-Density NVRAMs. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 710–723, https://ieeexplore.ieee.org/document/8574580, 2018. IEEE.
[18] J.Y. Wu, Y.S. Chen, W. S. Khwa, S. M. Yu, T. Y. Wang, J.C. Tseng, Y.D. Chih, and Carlos H. Diaz. A 40nm Low-Power Logic Compatible Phase Change Memory Technology. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 27.6.1–27.6.4, https://ieeexplore.ieee.org/abstract/document/8614513, 2018. IEEE.
[19] Sanghyeon Lee, Gwihyun Kim, Seungwoo Hong, Seung Jae Baik, Hideki Hori, and Dong-ho Ahn. Enhanced cycling endurance in phase change memory via electrical control of switching induced atomic migration. In *2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS)*, pages 1–3, https://ieeexplore.ieee.org/document/7060866, 2014. IEEE.
[20] H. Y. Cheng, T. H. Hsu, S. Raoux, J.Y. Wu, P. Y. Du, M. Breitwisch, Y. Zhu, E. K. Lai, E. Joseph, S. Mittal, R. Cheek, A. Schrott, S. C. Lai, H. L. Lung, and C. Lam. A high performance phase change memory with fast switching speed and high temperature retention by engineering the GexSbyTez phase change material. In *2011 International Electron Devices Meeting*, pages 3.4.1–3.4.4, https://ieeexplore.ieee.org/document/6131481, 2011. IEEE.
[21] H. Y. Cheng, M. BrightSky, S. Raoux, C. F. Chen, P. Y. Du, J. Y. Wu, Y. Y. Lin, T. H. Hsu, Y. Zhu, S. Kim, C. M. Lin, A. Ray, H. L. Lung, and C. Lam. Atomic-level engineering of phase change material for novel fast-switching and high-endurance PCM for storage class memory application. In *2013 IEEE International Electron Devices Meeting*, pages 30.6.1–30.6.4, https://ieeexplore.ieee.org/document/6724726, 2013. IEEE.
[22] Y. Hayakawa, A. Himeno, R. Yasuhara, W. Boullart, E. Vecchio, T. Vandeweyer, T. Witters, D. Crotti, M. Jurczak, S. Fujii, S. Ito, Y. Kawashima, Y. Ikeda, A. Kawahara, K. Kawai, Z. Wei, S. Muraoka, K. Shimakawa, T. Mikawa, and S. Yoneda. Highly reliable TaOx ReRAM with centralized filament for 28-nm embedded application. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*, pages T14–T15, https://ieeexplore.ieee.org/document/7231381, 2015. IEEE.
[23] Zixuan Chen, Huaqiang Wu, Bin Gao, Dong Wu, Ning Deng, He Qian, Zhichao Lu, Brent Haukness, M. Kellam, and Gary Bronner. Performance Improvements by SL-Current Limiter and Novel Programming Methods on 16MB RRAM Chip. In *2017 IEEE International Memory Workshop (IMW)*, pages 1–4, https://ieeexplore.ieee.org/document/7939097, 2017. IEEE.
[24] Alessandro Grossi, Elisa Vianello, Mohamed M. Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q. Le, Mary K. Wootters, Cristian Zambelli, Etienne Nowak, and Subhasish Mitra. Resistive

RAM Endurance: Array-Level Characterization and Correction Techniques Targeting Deep Learning Applications. *IEEE Transactions on Electron Devices*, 66(3):1281–1288, March 2019.

[25] Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Ku-Feng Lin, Tien-Wei Chiang, Yuan-Jen Lee, Kuei-Hung Shen, Roger Wang, Wayne Wang, Harry Chuang, Eric Wang, Yu-Der Chih, and Jonathan Chang. A Reflow-capable, Embedded 8Mb STT-MRAM Macro with 9nS Read Access Time in 16nm FinFET Logic CMOS Process. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 11.4.1–11.4.4, https://ieeexplore.ieee.org/document/9372115, 2020. IEEE.

[26] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, Harry Chuang, and Tsung-Yung Jonathan Chang. 13.3 A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150℃ and High Immunity to Magnetic Field Interference. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 222–224, https://ieeexplore.ieee.org/document/9062955, 2020. IEEE.

[27] Intel. High Bandwidth Memory (HBM2) DRAM Bandwidth, 2023.

[28] Matt Poremba, Sparsh Mittal, Dong Li, Jeffrey S. Vetter, and Yuan Xie. DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches. In *2015 Design, Automation, & Test in Europe Conference Exhibition (DATE)*, pages 1543–1546, https://ieeexplore.ieee.org/document/7092634, 2015. IEEE.

[29] Sparsh Mittal, Matt Poremba, Jeffrey Vetter, and Yuan Xie. Exploring Design Space of 3D NVM and eDRAM Caches Using DESTINY Tool, 01 2015.

[30] Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. Memory-Systems Challenges in Realizing Monolithic Computers. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '18, page 98–104, New York, NY, USA, 2018. Association for Computing Machinery.

[31] Daniel Sanchez and Christos Kozyrakis. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. In *2013 ACM/IEEE 40th Annual International Symposium on Computer Architecture (ISCA)*, ISCA '13, page 475–486, New York, NY, USA, 2013. Association for Computing Machinery.

[32] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, page 41–42, New York, NY, USA, 2018. Association for Computing Machinery.

[33] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, https://arxiv.org/abs/1512.03385, 2016. IEEE.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, https://arxiv.org/pdf/1406.2661.pdf, 2014. ACM.

[38] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, 2015.

[39] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, USA, 2016. USENIX Association.

[40] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[41] Julian Shun and Guy E. Blelloch. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, page 135–146, New York, NY, USA, 2013. Association for Computing Machinery.

[42] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 44–54, New York, NY, USA, 2006. Association for Computing Machinery.

[43] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics*, 6(1):29 – 123, 2009.

[44] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446, https://epubs.siam.org/doi/10.1137/1.9781611972740.43, 2004. SIAM, SIAM.

[45] P.P. Chu and R. Gottipati. Write buffer design for on-chip cache. In *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 311–316, https://ieeexplore.ieee.org/document/331913, 1994. IEEE.

[46] Pierre Guironnet de Massas and Frederic Petrot. Comparison of memory write policies for NoC based Multicore Cache Coherent Systems. In *2008 Design, Automation and Test in Europe*, pages 997–1002, https://ieeexplore.ieee.org/abstract/document/4484811, 2008. IEEE.

[47] Wenlei Bao, Sriram Krishnamoorthy, Louis-Noel Pouchet, and P. Sadayappan. Analytical Modeling of Cache Behavior for Affine Programs. *Proc. ACM Program. Lang.*, 2(POPL), dec 2017.

[48] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 14–23, Austin TX USA, June 2009. ACM.

[49] Derek Bruening and Saman Amarasinghe. *Efficient, transparent, and comprehensive runtime code manipulation*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2004.

[50] DynamoRIO. drcachesim: Tracing and analysis framework, 2023.

[51] Red Hat. 5.2. Huge Pages and Transparent Huge Pages, 2023.

[52] TU Graz. Paging on Intel x86-64, 2023.

[53] Moinuddin K. Qureshi and Gabe H. Loh. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 235–246, https://ieeexplore.ieee.org/document/6493623, 2012. IEEE.

[54] Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 25–37, https://ieeexplore.ieee.org/document/7011375, 2014. IEEE.

[55] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, https://ieeexplore.ieee.org/document/4919648, 2009. IEEE.

[56] Akshay Jain, Mahmoud Khairy, and Timothy G. Rogers. A Quantitative Evaluation of Contemporary GPU Simulation Methodology. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(2), jun 2018.

[57] Md Aamir Raihan, Negar Goli, and Tor Aamodt. Modeling deep learning accelerator enabled gpus, 2019.

[58] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 473–486, https://ieeexplore.ieee.org/document/9138922, 2020. IEEE.

[59] NVIDIA. Data Sheet: Quadro GV100, 2022.

[60] TechPowerUp. NVIDIA Tesla V100 PCIe 16 GB specs, 2023.

[61] NVIDIA. nvidia/cutlass: CUDA Templates for Linear Algebra Subroutines, 2022.

[62] Baidu. baidu-research/DeepBench: Benchmarking Deep Learning operations on different hardware, 2023.

[63] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127:29, 2012.

[64] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, https://ieeexplore.ieee.org/document/5306797, 2009. IEEE.

[65] Shuai Che, Jeremy W. Sheaffer, Michael Boyer, Lukasz G. Szafaryn, Liang Wang, and Kevin Skadron. A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads. In *IEEE International Symposium on Workload Characterization (IISWC'10)*, pages 1–11, https://ieeexplore.ieee.org/document/5650274, 2010. IEEE.

[66] Louis-Noël Pouchet. PolyBench/C: the Polyhedral Benchmark suite, 2023.

[67] Jonathan M. Smith. A Survey of Process Migration Mechanisms. *SIGOPS Oper. Syst. Rev.*, 22(3):28–40, jul 1988.

[68] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process Migration. *ACM Comput. Surv.*, 32(3):241–299, sep 2000.

[69] Hiroyuki Takizawa, Kentaro Koyama, Katsuto Sato, Kazuhiko Komatsu, and Hiroaki Kobayashi. CheCL: Transparent Checkpointing and Process Migration of OpenCL Applications. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 864–876, https://ieeexplore.ieee.org/document/6012895, 2011. IEEE.

[70] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual

Machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, USA, 2005. USENIX Association.

[71] Chao Wang, F. Mueller, C. Engelmann, and S.L. Scott. Proactive process-level live migration in HPC environments. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Austin, TX, November 2008. IEEE.

[72] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM international symposium on High performance distributed computing - HPDC '09*, page 101, Garching, Germany, 2009. ACM Press.

[73] Yuyang Du, Hongliang Yu, Guangyu Shi, Jian Chen, and Weimin Zheng. Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines. In *2010 39th International Conference on Parallel Processing*, pages 141–149, San Diego, CA, USA, September 2010. IEEE.

[74] Bolin Hu, Zhou Lei, Yu Lei, Dong Xu, and Jiandun Li. A Time-Series Based Precopy Approach for Live Migration of Virtual Machines. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 947–952, Tainan, Taiwan, December 2011. IEEE.

[75] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing - HPDC '11*, page 171, San Jose, California, USA, 2011. ACM Press.

[76] Yangyang Wu and Ming Zhao. Performance Modeling of Virtual Machine Live Migration. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 492–499, Washington, DC, USA, July 2011. IEEE.

[77] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. Towards unobtrusive VM live migration for cloud computing platforms. In *Proceedings of the Asia-Pacific Workshop on Systems - APSYS '12*, pages 1–6, Seoul, Republic of Korea, 2012. ACM Press.

[78] Wenjin Hu, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, and Jeanna N. Matthews. A quantitative study of virtual machine live migration. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference on - CAC '13*, page 1, Miami, Florida, 2013. ACM Press.

[79] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. VM Live Migration At Scale. *ACM SIGPLAN Notices*, 53(3):45–56, December 2018.

[80] George Michelogiannakis, Madeleine Glick, John Shalf, and Keren Bergman. Photonics as a means to implement intra-rack resource disaggregation. In Atul K. Srivastava, Madeleine Glick, and Youichi Akasaka, editors, *Metro and Data Center Optical Networks and Short-Reach Links V*, volume 12027, pages 64 – 73, https://doi.org/10.1117/12.2607317, 2022. International Society for Optics and Photonics, SPIE.

[81] George Michelogiannakis, Benjamin Klenk, Brandon Cook, Min Yee Teh, Madeleine Glick, Larry Dennison, Keren Bergman, and John Shalf. A Case For Intra-Rack Resource Disaggregation in HPC. *ACM Trans. Archit. Code Optim.*, 19(2), mar 2022.

[82] Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *2007 IEEE International Symposium on Circuits and Systems*, pages 3014–3017, New Orleans, LA, May 2007. IEEE.

[83] Sangyeun Cho and Hyunjin Lee. Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–357, New York New York, December 2009. ACM.

[84] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. PDRAM: a hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Annual Design Automation Conference*, pages 664–469, San Francisco California, July 2009. ACM.

[85] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, Austin TX USA, June 2009. ACM.

[86] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, New York New York, December 2009. ACM.

[87] Alexandre P Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mosse. Increasing PCM main memory lifetime. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 914–919, Dresden, March 2010. IEEE.

[88] Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10*, page 383, Saint-Malo, France, 2010. ACM Press.

[89] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page placement in hybrid memory systems. In *Proceedings of the international conference on Supercomputing*, pages 85–95, Tucson Arizona USA, May 2011. ACM.

[90] Chi-Hao Chen, Pi-Cheng Hsiu, Tei-Wei Kuo, Chia-Lin Yang, and Cheng-Yuan Michael Wang. Age-based PCM wear leveling with nearly zero search cost. In *Proceedings of the 49th Annual Design Automation Conference*, pages 453–458, San Francisco California, June 2012. ACM.

[91] Inhwan Choi and Dongkun Shin. Wear Leveling for PCM Using Hot Data Identification. In Kuinam J. Kim and Seong Jin Ahn, editors, *Proceedings of the International Conference on IT Convergence and Security 2011*, pages 145–149, Dordrecht, 2012. Springer Netherlands.

[92] Duo Liu, Tianzheng Wang, Yi Wang, Zhiwei Qin, and Zili Shao. A block-level flash memory management scheme for reducing write activities in PCM-based embedded systems. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1447–1450, Dresden, March 2012. IEEE.

[93] Joosung Yun, Sunggu Lee, and Sungjoo Yoo. Bloom filter-based dynamic wear leveling for phase-change RAM. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1513–1518, Dresden, March 2012. IEEE.

[94] Dongki Kim, Sungkwang Lee, Jaewoong Chung, Dae Hyun Kim, Dong Hyuk Woo, Sungjoo Yoo, and Sunggu Lee. Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU. In *Proceedings of the 49th Annual Design Automation Conference*, pages 888–896, San Francisco California, June 2012. ACM.

[95] Tianzheng Wang, Duo Liu, Zili Shao, and Chengmo Yang. Write-activity-aware page table management for PCM-based embedded systems. In *17th Asia and South Pacific Design Automation Conference*, pages 317–322, Sydney, Australia, January 2012. IEEE.

[96] Duo Liu, Tianzheng Wang, Yi Wang, Zili Shao, Qingfeng Zhuge, and E. Sha. Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 279–284, Yokohama, January 2013. IEEE.

[97] Hoda Aghaei Khouzani, Chengmo Yang, and Jingtong Hu. Improving performance and lifetime of DRAM-PCM hybrid main memory through a proactive page allocation strategy. In *The 20th Asia and South Pacific Design Automation Conference*, pages 508–513, Chiba, Japan, January 2015. IEEE.

[98] Joosung Yun, Sunggu Lee, and Sungjoo Yoo. Dynamic Wear Leveling for Phase-Change Memories With Endurance Variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(9):1604–1615, September 2015.

[99] Lunkai Zhang, Brian Neely, Diana Franklin, Dmitri Strukov, Yuan Xie, and Frederic T. Chong. Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 519–531, Seoul, South Korea, June 2016. IEEE.

[100] Andrés Amaya García, René de Jong, William Wang, and Stephan Diestelhorst. Composing lifetime enhancing techniques for non-volatile main memories. In *Proceedings of the International Symposium on Memory Systems*, pages 363–373, Alexandria Virginia, October 2017. ACM.

[101] Mohammad Reza Jokar, Lunkai Zhang, and Frederic T. Chong. Cooperative NV-NUMA: prolonging non-volatile memory lifetime through bandwidth sharing. In *Proceedings of the International Symposium on Memory Systems*, pages 67–78, Alexandria Virginia USA, October 2018. ACM.

[102] Jie Xu, Dan Feng, Yu Hua, Wei Tong, Jingning Liu, Chunyan Li, and Zheng Li. An efficient PCM-based main memory system via exploiting fine-grained dirtiness of cachelines. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1616–1621, Dresden, Germany, March 2018. IEEE.

[103] Wei Li, Ziqi Shuai, Chun Jason Xue, Mengting Yuan, and Qingan Li. A Wear Leveling Aware Memory Allocator for Both Stack and Heap Management in PCM-based Main Memory Systems. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 228–233, Florence, Italy, March 2019. IEEE.

[104] Yifu Deng, Jianhui Yue, Zhiyuan Lu, and Yifeng Zhu. Efficient Hardware-assisted Out-place Update for Persistent Memory. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 507–512, Grenoble, France, February 2021. IEEE.

[105] Robert M. Radway, Andrew Bartolo, Paul C. Jolly, Zainab F. Khan, Binh Q. Le, Pulkit Tandon, Tony F. Wu, Yunfeng Xin, Elisa Vianello, Pascal Vivet, Etienne Nowak, H.-S. Philip Wong, Mohamed M. Sabry Aly, Edith Beigne, Mary Wootters, and Subhasish Mitra. Illusion of large on-chip memory by networked computing chips for neural network inference. *Nature Electronics*, 4(1):71–80, January 2021.

[106] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of PCM-based Main Memory with Start-Gap Wear Leveling. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 14–23, https://ieeexplore.ieee.org/document/5375309, 2009. IEEE.

[107] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 24–33, New York, NY, USA, 2009. Association for

Computing Machinery.

[108] Chi-Hao Chen, Pi-Cheng Hsiu, Tei-Wei Kuo, Chia-Lin Yang, and Cheng-Yuan Michael Wang. Age-Based PCM Wear Leveling with Nearly Zero Search Cost. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 453–458, New York, NY, USA, 2012. Association for Computing Machinery.

[109] Jishen Zhao and Yuan Xie. Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 81–87, https://ieeexplore.ieee.org/document/6386592, 2012. IEEE.

[110] Jangwoong Kim, Youngjae Kim, Awais Khan, and Sungyong Park. Understanding the performance of storage class memory file systems in the NUMA architecture. *Cluster Computing*, 22:347–360, 2018.

[111] Rino Micheloni, Alessia Marelli, and Kam Eshghi. *Inside Solid State Drives (SSDs)*. Springer, https://link.springer.com/book/10.1007/978-981-13-0599-3, 2013.

[112] Song-He Liu, Xiang-Mo Zhao, Jun Zhang, and Ya-Nan Huang. A Static Trigger Wear-Leveling Strategy for Flash Memory In Embedded System. In *2008 Fifth IEEE International Symposium on Embedded Computing*, pages 255–259, https://ieeexplore.ieee.org/document/4690758, 2008. IEEE.

[113] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. Improving Flash Wear-Leveling by Proactively Moving Static Data. *IEEE Transactions on Computers*, 59(1):53–65, 2010.

[114] Muthukumar Murugan and David.H.C. Du. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, https://ieeexplore.ieee.org/document/5937225, 2011. IEEE.

[115] Li-Pin Chang, Tung-Yang Chou, and Li-Chun Huang. An Adaptive, Low-Cost Wear-Leveling Algorithm for Multichannel Solid-State Disks. *ACM Trans. Embed. Comput. Syst.*, 13(3), dec 2013.

[116] Fu-Hsin Chen, Ming-Chang Yang, Yuan-Hao Chang, and Tei-Wei Kuo. PWL: A progressive wear leveling to minimize data migration overheads for NAND flash devices. In *2015 Design, Automation, & Test in Europe Conference & Exhibition (DATE)*, pages 1209–1212, https://ieeexplore.ieee.org/document/7092571, 2015. IEEE.

[117] Li-Pin Chang. On Efficient Wear Leveling for Large-Scale Flash-Memory Storage Systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, page 1126–1130, New York, NY, USA, 2007. Association for Computing Machinery.

[118] Dawoon Jung, Yoon-Hee Chae, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. A Group-Based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems. In *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '07, page 160–164, New York, NY, USA, 2007. Association for Computing Machinery.

[119] Li-Pin Chang and Chun-Da Du. Design and Implementation of an Efficient Wear-Leveling Algorithm for Solid-State-Disk Microcontrollers. *ACM Trans. Des. Autom. Electron. Syst.*, 15(1), dec 2010.

[120] Li-Pin Chang and Li-Chun Huang. A Low-Cost Wear-Leveling Algorithm for Block-Mapping Solid-State Disks. In *Proceedings of the 2011 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '11, page 31–40, New York, NY, USA, 2011. Association for Computing Machinery.

[121] Surafel Teshome and Tae-Sun Chung. A Tri-Pool Dynamic Wear-Leveling Algorithm for Large Scale Flash Memory Storage Systems. In *2011 International Conference on Information Science and Applications*, pages 1–6, https://ieeexplore.ieee.org/document/5772379, 2011. IEEE.

[122] Jianwei Liao, Fengxiang Zhang, Li Li, and Guoqiang Xiao. Adaptive Wear-Leveling in Flash-Based Memory. *IEEE Computer Architecture Letters*, 14(1):1–4, 2015.

[123] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 381–394, New York, NY, USA, 2013. Association for Computing Machinery.

[124] Jaydeep Marathe and Frank Mueller. Hardware Profile-Guided Automatic Page Placement for ccNUMA Systems. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '06, page 90–99, New York, NY, USA, 2006. Association for Computing Machinery.

[125] David Gureya, Joao Neto, Reza Karimi, Joao Barreto, Pramod Bhatotia, Vivien Quema, Rodrigo Rodrigues, Paolo Romano, and Vladimir Vlassov. Bandwidth-Aware Page Placement in NUMA. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 546–556, New Orleans, LA, USA, May 2020. IEEE.

[126] Jonathan Corbet. AutoNUMA: the other approach to NUMA scheduling, 2012.