CompEx: Compression-Expansion Coding for Energy, Latency, and Lifetime Improvements in MLC/TLC NVM

Poovaiah M. Palangappa and Kartik Mohanram
Department of Electrical and Computer Engineering, University of Pittsburgh, PA
pmp30@pitt.edu kartik.mohanram@gmail.com

Abstract

Multi-level/triple-level cell non-volatile memories (MLC/TLC NVMs) such as PCM and RRAM are the subject of active research and development as replacement candidates for DRAM, which is limited by its high refresh power and poor scaling potential. Besides the benefits of non-volatility (low refresh power) and improved scalability, MLC/TLC NVMs offer high data density and memory capacity over DRAM. However, the viability of MLC/TLC NVMs is limited primarily due to the high programming energy and latency as well as the low endurance of NVM cells; these are primarily attributable to the iterative program-and-verify procedure necessary for programming the NVM cells.

In this paper, we propose compression-expansion (CompEx) coding, a low overhead scheme that synergistically integrates statistical compression with expansion coding to realize simultaneous energy, latency, and lifetime improvements in MLC/TLC NVMs. CompEx coding is agnostic to the choice of compression technique; in this paper, we evaluate CompEx coding using both frequent pattern compression (FPC) and base-delta-immediate (BΔI) compression. CompEx coding integrates FPC/BΔI with $(k,m)_q$ 'expansion' coding; expansion codes are a class of q-ary linear block codes that encode data using only the low energy states of a q-ary NVM cell. CompEx coding simultaneously reduces energy and latency and improves lifetime for no memory overhead and negligible logic overhead ($\approx$ 10k gates, which is $<$ 0.1% per NVM module). Our full-system simulations of a system that integrates TLC RRAM show that CompEx coding reduces total memory energy by 53% and write latency by 24%; these improvements translate to a 5.7% improvement in IPC, a 11.8% improvement in main memory bandwidth, and 1.8$\times$ improvement in lifetime over classical binary coding using data-comparison write. CompEx coding thus addresses the programming energy/latency and lifetime challenges of MLC/TLC NVMs that pose a serious technological roadblock to their adoption in high performance computing systems.

1. Introduction

The ITRS projects that resistance-class non-volatile memory (NVM) technologies such as phase-change memory (PCM) [1] and resistive RAM (RRAM) [2] are suitable replacement candidates for DRAM due to their low refresh power and better scaling potential [3]. Resistance-class NVMs store data by modulating the resistance of the storage material. Due to the wide range in the resistance of the NVM cells, multi-level/triple-level cell NVMs (MLC/TLC NVMs) that can be programmed to more than two distinct resistance ranges are the subject of active research and development [4]. MLC/TLC NVMs offer higher data density in comparison to single-level cell NVMs (SLC NVMs) since they can store more than 1 logical data-bit. However, MLC/TLC NVMs aggravate the challenges of working with SLC NVMs, primarily due to (i) increased resistance drift, which significantly diminishes non-volatility [5], and (ii) the complex, iterative program-and-verify procedure necessary to program MLC/TLC cells [6], which increases access latencies as well as power/energy requirements [7] and reduces cell lifetime due to high currents and multiple iterations on writes [8].

Problem: Whereas MLC/TLC NVMs offer higher capacity by packing multiple logical bits/cell, the energy and latency to program each cell is high in comparison to SLC NVMs. Figure 1(a) shows the energy and latency v/s capacity trends for SLC, MLC, and TLC RRAM [9]. This increase in energy and latency for programming an MLC/TLC NVM cell translates to system level performance penalties in terms of IPC, bandwidth, and energy as shown in Fig. 1(b), which is measured for RRAM by using the Stream benchmark suite [10] on MARSS [11] and DRAMSim2 [12]. These challenges of MLC/TLC NVMs pose a serious technological roadblock to their adoption in high performance computing systems, which is the core focus of this paper.

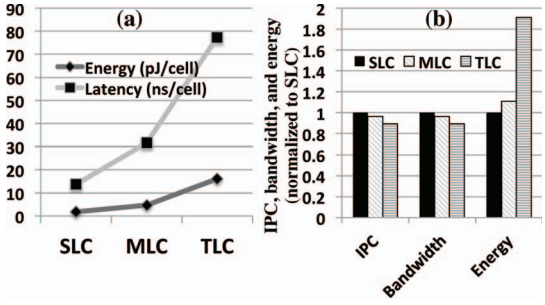

Figure 1: **(a) Programming energy and latency trends for SLC, MLC, and TLC RRAM [9]. (b) Performance penalties for MLC/TLC RRAM in comparison to SLC RRAM for the Stream benchmarks [10] using MARSS [11] and DRAMSim2 [12]. The key take-away is that the move to denser technologies like MLC/TLC usually degrades IPC, bandwidth, and energy.**

Related work and motivation: MLC/TLC NVMs definitely benefit from the broad set of solutions developed to improve energy, latency, and lifetime of SLC NVMs in the literature. Example solutions include coding [13–16], write scheduling [17–20], data-migration using address translation [21–23], and architectural improvements [24–26]. However, these solutions do not address the energy/latency problems of MLC/TLC NVMs that is primarily due to the iterative nature of

This research was supported in part by NSF CAREER Award CCF-1208933 and in part by NSF Award CCF-1217738.

978-1-4673-9211-2/16/$31.00 © 2016 IEEE

programming a cell. Solutions that explicitly address the challenges of working with MLC/TLC NVMs have mostly focused on data compression and data coding. Whereas data compression techniques such as [26–34] approach this problem by reducing the number of cell-writes per write access, data encoding solutions such as [5,8,9,35–37] reduce energy and latency by using only the desirable states and excluding undesirable states of an MLC/TLC NVM cell. However, these solutions are limited by computation, memory, and logic overheads. This motivates the development of solutions that simultaneously realize the advantages of both data compression and data coding for no memory overhead.

Contributions: This paper proposes compression-expansion (CompEx) coding, a low overhead scheme that synergistically integrates statistical compression with expansion coding to realize simultaneous energy and latency improvements in MLC/TLC NVMs. The core idea of CompEx coding is to ***selectively apply expansion codes***, i.e., linear block codes that encode data using only the low energy states of an MLC/TLC cell ***to compressed data***, thereby ensuring that the resulting data in expansion-coded form will not exceed the original data width. Thus, CompEx coding can translate to energy, latency, and lifetime improvements in MLC/TLC NVMs for no memory overhead (although CompEx coding requires a tag bit, this is absorbed at no cost for data widths that are powers of 2), which is a significant advancement over solutions in the literature. Although CompEx coding is agnostic to the choice of compression technique, in this work, CompEx coding is evaluated using two compression techniques — frequent pattern compression (FPC) [38] and base-delta-immediate (BΔI) compression [39]. FPC is a low overhead, lossless word-level compression technique that uses a static pattern table to match a wide range of values without the need for offline/online application profiling. BΔI is a compression technique that is similar to FPC in principle, but operates at cache line level instead of word-level. BΔI takes advantage of regularity of data storage and the limited dynamic range of stored data. Following compression, the compressed data is selectively encoded using $(k,m)_q$ expansion coding. Expansion codes are a class of q-ary linear block codes that encode data using only p lowest energy/latency cell states ($p < q$). This paper evaluates two examples of CompEx coding — $(3,2)_8$ and $(6,5)_8$ expansion coding. Expansion codes are studied theoretically in the context of MLC/TLC NVMs and theoretical estimates for the maximum achievable energy reductions for MLC/TLC NVMs are derived. The expected energy reduction for $(3,2)_8$ and $(6,5)_8$ expansion coding is 2.3× and 1.4×, respectively, over classical binary encoding; in close agreement with these estimates, our results on the SPEC CPU-2006 [40] benchmark suite show that it is possible to realize energy reductions of 2.2× and 1.25× in practice.

Evaluation: In this work, the efficiency of CompEx coding is evaluated on a 64-bit word, 64-byte cache line architecture, i.e., every word is stored using 22 ($\lceil 64/3 \rceil$) TLCs and every cache line is stored using 171 ($\lceil 512/3 \rceil$) TLCs. CompEx coding relies on a codec embedded in the NVM module controller, which abstracts all data manipulations from the memory controller on the processor side, allowing the processor to communicate seamlessly with the NVM module. On memory writes, the CompEx codec attempts to compress the data at the word-level using FPC or the cache-line level using BΔI. If compression is successful, the data is encoded using $(3,2)_8$ expansion coding before it is written into the NVM array; else the data is written as-is into the NVM array. Note that although a single tag bit is necessary to record the outcome of CompEx coding, it is concatenated with the data (513 logical bits for BΔI and 65 logical bits for FPC) and absorbed into the last TLC at no cost ($\lceil 513/3 \rceil = \lceil 512/3 \rceil = 171$ TLCs for BΔI and $\lceil 65/3 \rceil = \lceil 64/3 \rceil = 22$ TLCs for FPC). On memory reads, the tag bit is recovered from the last TLC and used to determine if the word must be decoded using the CompEx codec, or if it can be forwarded directly to the NVM module controller. The logic overhead of the CompEx codec[1] is ≈ 10k gates (< 0.1% per NVM module); although the codec has a latency of 2-3 cycles, we show that this can be hidden through memory access pipelining in practice.

Results: Our full-system simulations of a system that integrates TLC RRAM using MARSS [11] and DRAMSim2 [12] for workloads from the SPEC CPU2006 [40] benchmark suite show that CompEx coding improves system performance by 5.7%, as measured using IPC, and memory-system performance by 11.8%, as measured using memory bandwidth, in comparison to binary encoding using data-comparison write (DCW) [14] (a read-modify-write process that only updates changed cells). For a deep, multi-billion-instruction evaluation of CompEx coding, we use NVMain [41] with the memory traces of SPEC CPU2006 benchmarks generated using the Intel Pin toolset [42]. Simulations of these traces show that CompEx coding reduces total (dynamic) write energy by 53% (74%), 20% (21%), and 19% (21%) on average over DCW, FPC, and BΔI, respectively. Simultaneously, CompEx coding reduces write latency by 28%, 17%, and 20% on average in comparison to DCW, FPC, and BΔI, respectively. Finally, we show that FPC-based CompEx coding improves TLC RRAM lifetime by 1.8×.

This paper is organized as follows. Section 2 provides the background for energy/latency problems in MLC/TLC NVMs. Section 3 describes CompEx coding with examples. Section 4 presents the evaluation methodology and the simulation setup. Section 5 presents evaluation results. Section 6 is related work and Section 7 presents conclusions.

2. Background and motivation

MLC/TLC NVM technologies such as PCM and RRAM are the subject of active research and development as replacement candidates for DRAM, which is limited by its high refresh power and poor scaling potential [43–46]. However, most MLC/TLC NVMs suffer from higher write energy [7, 46] and latency [47] per cell, limited lifetime [1] and asymmetric read/write latencies [7, 25, 48–50]. In this section, we discuss the procedure used for programming an MLC/TLC NVM cell and its associated limitations.

MLC/TLC NVM program-and-verify (P&V): MLC/TLC NVM exhibits non-deterministic behavior due to process variation, variation in material composition, and resistance drift. This non-determinism increases the complexity of program-

[1] In this work, only the logic overhead of the FPC-based CompEx codec is evaluated. The logic overhead of the BΔI-based CompEx codec is based on the original proposal of classical BΔI [39].

ming an MLC/TLC NVM to the right range of resistance using a single precise pulse of current or voltage. Therefore, in practice, MLC/TLC NVMs are programmed using an iterative P&V procedure to bring the resistance of the cell to the required range. P&V uses a combination of *set-to-reset* (STR) and *reset-to-set* (RTS). In STR (RTS), the cell is first brought to a full SET (RESET) state by applying a long (short) pulse of small (high) magnitude current. Following this, multiple small duration RESET (SET) pulses are applied until the resistance of the cell is brought to the required range. However, since a TLC NVM has 8 different resistance states, using only STR or RTS leads to high write latency. Hence, TLC NVMs are programmed using a combination of both STR and RTS depending on the proximity of the target state to the full SET/RESET states as shown in Fig. 2. Therefore, the write energy and latency for MLC/TLC NVM can vary depending on the final state to which the cell is being programmed. For example, Fig. 2 shows the average energy and latency for programming a TLC RRAM into different states [9]. The states 0 and 7 require the lowest energy and latency since they can be programmed using a single P&V iteration, whereas the states 3 and 4 require maximum energy and latency since they require the most number of P&V iterations to be brought to the desired resistance range.

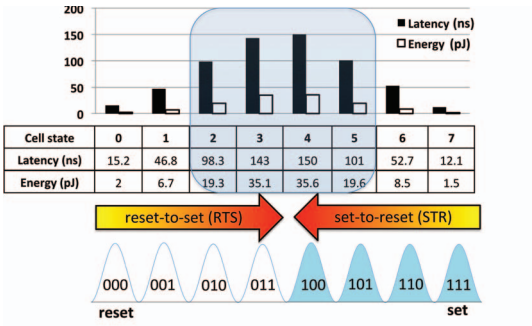

Figure 2: **Energy and latency for TLC RRAM [9] using P&V across 8 states. States 0-3 and 4-7 are programmed using RTS and STR, respectively. Programming the central states (2, 3, 4, and 5) requires more energy and latency in comparison to the terminal states (0, 1, 6, and 7). The key take-away is that the overall energy and latency can be reduced by encoding data using only the low energy states.**

Impact of P&V on energy and latency: Iterative P&V has a negative impact on the energy and latency of MLC/TLC NVMs. First, writing to MLC/TLC NVMs require much higher current and latency than their SLC counter parts due to the iterative P&V method. Second, in order to prevent the modules from overloading the power supply, and to prevent overheating, NVM modules are restricted to writing a limited amount of data at once, which increases the write latency, impacting performance [25]. Third, since a single write access can potentially require MLC/TLCs to be brought to different target states in a word, the latency of the write operation is determined by the longest latency cell write, creating a bottleneck for individual write operations [7]. Finally, a single write for MLC/TLC NVMs involves multiple P&V cycles, which limits the lifetime of these memory cells to around $10^{5–8}$ writes [7] in comparison to SLC NVM, which has a lifetime of $10^{8–10}$ writes [1, 45, 46].

3. Contributions

Moving to an MLC/TLC technology from an SLC technology (and similarly to a TLC technology from an MLC technology) enables static tradeoffs between memory density/capacity and memory energy/latency. We motivated the technology considerations and the system level implications of these tradeoffs in Fig. 1. However, to the best of our knowledge, there has been no systematic effort to dynamically leverage these tradeoffs to realize the density/capacity benefits of an MLC NVM (TLC NVM) while also reaping the low energy/latency benefits of an SLC NVM (SLC/MLC NVM). In other words, we believe that the ability to integrate a dense high capacity MLC/TLC technology for the memory while dynamically operating that memory in the SLC/MLC mode (fully or partially, as developed in the theory of expansion codes in this section) has far-reaching implications for simultaneously improving NVM energy, latency, density/capacity, and lifetime (the lifetime improvement is indirect, due to a reduction in the P&V effort needed for memory operation). CompEx coding, as one instance of such a dynamic tradeoff approach, relies on the holistic integration of two independent/orthogonal but complementary areas of research (data compression and data coding). The core idea of CompEx coding, developed over the rest of this section, is to apply expansion coding to selectively compressed data such that the resulting data in expansion coded form does not exceed the original data width.

Section 3.1 introduces two statistical compression techniques that can be used in CompEx coding — frequent pattern compression [38] and base-delta-immediate compression [39]. Section 3.2 provides a formal treatment of expansion coding. Section 3.3 describes CompEx coding from first principles with examples and discusses a practical architecture for CompEx coding.

3.1 Compression

Although CompEx coding is agnostic to the choice of compression technique, good candidates for CompEx coding should have features such as low latency, low overhead, high compressibility, and low complexity. Based on our survey of compression techniques, two compression techniques — frequent pattern compression and base-delta-immediate compression — possess these desirable traits for integration into CompEx coding.

Frequent pattern compression (FPC): FPC is a compression scheme that leverages program data statistics to successfully compress a wide range of data. FPC was originally proposed for 32-bit data word compression in L2 caches to increase their memory capacity [38]. More recently, FPC was extended to reduce bit-writes in NVMs [26]. In this work, we extend FPC to a 64-bit word using the patterns tabulated in Table 1. FPC maintains a table of seven most frequent data patterns, against which the incoming data is compared. When the incoming data matches one of the frequent patterns, it is compressed and stored along with a 3-bit prefix corresponding to the pattern which is represented by the column 1 of Table 1. A 1-bit tag is used to indicate whether the written data is its compressed/un-compressed form. During a read access, the tag bit and prefix is used to uncompress the data to a full word. Columns 3, 4, and 5 of Table 1 shows examples for each of the data patterns along with their com-

Prefix	Pattern encoded	Example	Compressed example	Encoded size	Value space	Compression contribution
000	Zero run	0x0000000000000000	0x0	3 bits	1 value	29.7%
001	8-bit sign-extended	0x000000000000007F	0x17F	11 bits	255 values	1%
010	16-bit sign-extended	0xFFFFFFFFFFFFB6B6	0x2B6B6	19 bits	65280 values	1%
011	Half-word sign-extended	0x0000000076543210	0x376543210	35 bits	$\approx 2^{32}$ values	2.1%
100	Half-word, padded with a zero half-word	0x7654321000000000	0x476543210	35 bits	$\approx 2^{32}$ values	16.4%
101	Two half-words, each a byte sign-extended	0xFFFFBEEF00003CAB	0x5BEEF3CAB	35 bits	$\approx 2^{32}$ values	8.3%
110	Word consisting of four repeated double bytes	0xCAFECAFECAFECAFE	0x6CAFE	19 bits	65534 values	1.1%

Table 1: **64-bit FPC patterns (inclusive of 3-bit prefix, indicated in red). Illustrative examples are given in columns 3 and 4. The last column shows the percentage of words that are compressed using a given pattern. On the whole, FPC can compress about 60% of write accesses for the benchmarks from SPEC CPU2006 [40] benchmark suite.**

pressed form and compressed data size. Column 6 of Table 1 represents the range of data values that each pattern can compress and column 7 represents the percentage of data words compressed by each pattern (FPC cumulatively compresses about 60% of write accesses) in trace-based simulations of the SPEC CPU2006 [40] benchmark suite.

Base-Delta-Immediate (BΔI): BΔI was originally proposed for on-chip cache data compression [39] by storing the compressed data using a 'base' (B) and 'delta' (Δ), a series of offsets with respect to B. BΔI proposes that a cache-line $C = \{V_0, V_1, ...V_{n-1}\}$ can be compressed and written as $X = \{B, \Delta_0, \Delta_1, ...\Delta_{n-1}\}$[2]. along with a 4-bit tag, where $B = V_0$ (definition), $\Delta_i = V_i - B$, n = sizeof(cache-line)/k, and k = sizeof(Δ). It is reported that data stored in a cache line is often regular, with limited dynamic range [39]. Whereas the regularity in the data is due to the common use of array data structures to store program data, the limited dynamic range in the stored data is due to the nature of computation. Similar to FPC, BΔI maintains a pattern table for compression. BΔI takes advantage of regularity in data and its limited range to compress cache lines using 64-byte patterns. Table 2 shows the 8 different cache line patterns that BΔI is capable of compressing, along with illustrative examples for each of the patterns. The last column represents the percentage of data words compressed by each pattern (BΔI cumulatively compresses about 46% of write accesses) in trace-based simulations of the SPEC CPU2006 [40] benchmark suite. Given a cache line, it is matched against each row (pattern) of Table 2. If the cache-line data matches a pattern, then the compressed data along with the prefix is stored in the data memory; the tag bit is set to indicate that the data stored is in compressed format. In contrast, if the data is not compressed, then it is stored as-is in the uncompressed format; the tag bit is reset to record this information.

Due to the low overhead and high compression ratio of FPC and BΔI, this work is motivated by the potential of integrating one of these compression techniques with expansion coding for a holistic solution that reduces energy and latency without any additional memory overhead.

3.2 Expansion coding

As described in Section 2, the iterative P&V procedure used for programming MLC/TLC NVMs results in the central MLC/TLC states requiring more energy and latency in comparison to the terminal states. This motivates data encoding using only the low energy states, avoiding the high energy states to reduce the overall energy and latency. This work uses *expansion coding*, which encodes data using only these lower energy states. The rest of this sub-section provides formal definitions for expansion coding and illustrates expansion coding with examples.

Definition: A $(k,m)_q$ expansion code, $m < k$, is a linear block code with q-ary code words of length k encoding q^m q-ary message words. Expansion code encodes data using p lowest energy states of total q states ($p < q$), where $p = \lceil q^{(m/k)} \rceil$, incuring memory overhead of at least $(k/m) - 1$ [51, Ch. 3].

We elaborate on the $(k,m)_q$ expansion code with the help of Fig. 3. Given a technology where each cell can represent q states, we can encode $\lfloor \log_2 q \rfloor$ bits of data in each cell. The total number of cells in each message is m. Hence each message word is one of the q^m possible words, where each word is q-ary. This is represented using the left-hand-side of the Fig. 3. All these messages are mapped to k cells, where each cell can represent only p states logically (though physically they are still capable of storing q states). This is represented using the right-hand-side of the Fig. 3. The code expands a m-cell word to a k-cell word after encoding, and hence the name 'expansion code'. Expansion coding lowers both NVM energy and latency in practice, since the lowest energy TLC states also require fewer P&V cycles (Fig. 2).

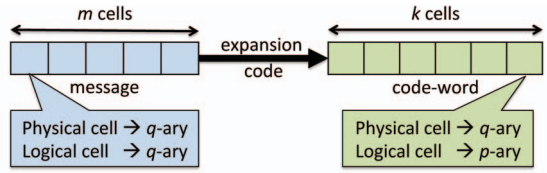

Figure 3: **Illustration of $(k,m)_q$ expansion coding. An m-cell q-ary message is encoded to a k-cell p-ary ($p < q$) message.**

Example: For ease of understanding, we illustrate $(3,2)_8$ expansion coding and IDM [9], which is an instance of $(6,5)_8$ expansion coding. First, consider an example of $(3,2)_8$ expansion code, which encodes 16-bit data using TLC RRAM as shown in Fig. 4. In the regular case, data is encoded using all 8 TLC states as shown in 4(a). Every slice of 3 logical bits of data is stored in one TLC, requiring 6 ($=\lceil 16/3 \rceil$) TLCs and 120.1pJ. In contrast, the $(3,2)_8$ expansion code uses only 4 ($\lceil 8^{(2/3)} \rceil$) out of the 8 TLC states to encode the data (Fig. 4(b)). Every slice of 2 logical bits of data is stored in one TLC, requiring 8 ($=\lceil 16/2 \rceil$) TLCs and 39.2pJ. In this example, although expansion coding results in en-

[2]This work integrates BΔI with a single 'base', without the implicit second base, for simplicity. This is similar to the base+delta technique proposed in the same paper [39]. Furthermore, instead of using a 4-bit tag as proposed, we use a single tag bit to indicate whether the stored data is in compressed (tag=1) or uncompressed (tag=0) form. However, a 3-bit prefix (only for compressed data) is stored along with the compressed line to indicate the BΔI pattern that was used for compression.

Prefix	Pattern	Example	Compressed example	Encoded size*	Compression contribution
000	Zeros	0x000000000000 000000000000 ... 000000000000	0x0	0 bytes	24.7%
001	Rep. values	0xFEEDC0FFEEC0FFEE FEEDC0FFEEC0FFEE ... FEEDC0FFEEC0FFEE	0x1FEEDC0FFEEC0FFEE	8 bytes	6.8%
010	Base8-Δ1	0xABCDABCDABCDAB00 ABCDABCDABCDAB01 ... ABCDABCDABCDAB08	0x2ABCDABCDABCDAB00,01,...,08	15 bytes	0.2%
011	Base8-Δ2	0xABCDABCDABCD0000 ABCDABCDABCD00FA ... ABCDABCDABCD5A5A	0x3ABCDABCDABCD0000,00FA,...,5A5A	22 bytes	0.2%
100	Base8-Δ4	0xABCDABCD00000000 ABCDABCDBEEFFEED ... ABCDABCDC0FFEEDC	0x4ABCDABCD00000000, BEEFFEED,...,C0FFEEDC	36 bytes	1.3%
101	Base4-Δ1	0xABCDAB00 ABCDAB01 ABCDAB02 ... ABCDABFF	0x5ABCDAB00,01,02,...,FF	19 bytes	3.2%
110	Base4-Δ2	0xABCD0000 ABCDFEED ABCDF00F ... ABCDEEDC	0x6ABCD0000,FEED,F00F,...,EEDC	34 bytes	6.6%
111	Base2-Δ1	0xAB00 ABFF ABED AB0F ... ABDC	0x7AB00,FF,ED,0F,...,DC	35 bytes	2.7%

* First, the encoded size is 3 bits in addition to this column's contents (for prefix). Second, since B=V_0 by definition, storing Δ_0 is redundant and hence omitted.

Table 2: **64-byte BΔI patterns (with 3-bit prefix, indicated in red) along with illustrative examples and encoded size post-compression. On the whole, BΔI can compress about 46% of write accesses for the benchmarks from SPEC CPU2006 [40] benchmark suite. Note that although the compression contributions of FPC and BΔI are comparable, the compression ratios are very different. For example, the bulk of the compression from 'Zero values' in both FPC and BΔI have comparable compression contributions. Whereas FPC compresses data from 64 bits to 3 bits, BΔI compresses 64 bytes (512 bits) to 3 bits. Hence we expect to see higher reduction in energy using BΔI in comparison to FPC. On the other hand, since latency is limited by the worst case cell-write delay, we expect latency improvement using BΔI to be comparable to that of FPC.**

ergy and latency reduction, it incurs 50% memory overhead. IDM [9] seeks to lower the memory overhead to 20% by using the $(6,5)_8$ expansion code utilizing the 6 lowest energy TLC states. Whereas binary coding encodes 3 bits of information into each TLC, IDM encodes 5 bits of data into 2 TLCs. Since 2 TLCs together can store a maximum 6×6=36 states, they can easily encode the 2^5=32 states required for storing information from 5-bits. For a 16-bit data encoded using IDM (Fig. 4 (c)), binary encoding uses 6 TLCs to represent 16 bits and requires 120.1pJ. However, using IDM, 16 bits of data can be encoded using 7 TLCs (using 6 states) and 94.3pJ.

Although IDM in combination with dynamic data remapping has been shown to improve the lifetime of MLC/TLC NVMs [9], it provides marginal reduction in energy (up to 15%) for 20% memory overhead. In this work, we look to data statistics as a potential source of flexibility to realize the full energy and latency benefits of expansion coding for negligible memory and logic overhead.

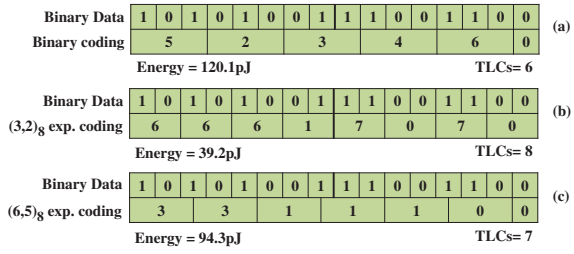

Figure 4: **Comparison of binary, $(3,2)_8$ expansion coding, and $(6,5)_8$ expansion coding (IDM [9]). Energy for $(3,2)_8$ expansion coding < $(6,5)_8$ expansion coding < binary coding.**

3.3 CompEx coding

The core idea of this paper is to apply expansion coding selectively to compressed data such that the resulting data in expanded form does not exceed the original data width. For example, if n-bit raw data is compressed to $\leq 2n/3$ bits, the $(3,2)_8$ expansion code will incur no memory overhead and also yield the full energy and latency benefits of expansion coding. However, in order to be successful in practice, the compression scheme must compress a large fraction of the memory traffic while also being lossless and simple to design with low performance overhead. FPC and BΔI are excellent candidates for this purpose due to their ability to compress a wide range of data for low overhead (refer Tables 1 and 2). Furthermore, since the largest size of an FPC-compressed 64-bit word (or BΔI-compressed 64-byte cache line) is only 19 bits (36 bytes), we can easily layer $(3,2)_8$ expansion coding atop FPC without incurring additional memory overhead. In this work, all examples and evaluation of CompEx coding use the $(3,2)_8$ expansion code (and the $(6,5)_8$ code as necessary/appropriate).

Example: Without loss of generality, we illustrate CompEx coding using 16-bit word as an example, which is compressed to 8 bits using FPC as shown in Fig. 5. In the first case, we encode the data using all the 8 TLC states from Fig. 2(a). This allows us to encode data using just 3 of the 6 TLCs in the memory location for 97.0pJ. However, using CompEx coding, as shown in Fig. 5(b), the 8-bit compressed data can be encoded using the $(3,2)_8$ expansion code. This uses 4 of the 6 TLCs for 24.9pJ. This example illustrates the integration of FPC with expansion coding without any additional memory overhead. Therefore, for a 64-bit word, CompEx coding encodes an FPC-compressed 64-bit word to a maximum of 18 ($\lceil 35/2 \rceil$) TLCs. Similarly Compex coding encodes a BΔI-compressed 64-byte cache-line to a maximum of 146 ($\lceil (36\times 8+4)/2 \rceil$) TLCs.

No tag overhead: Although a single tag bit is necessary to record the outcome of CompEx coding, it is concatenated with the data (65 bits for FPC-based and 513 bits for BΔI-based Compex codec) and absorbed into the last TLC at no cost ($\lceil 65/3 \rceil$ = 22 TLCs for FPC and $\lceil 513/3 \rceil$ = 171 TLCs for BΔI). Consider the 16-bit example shown in Fig. 5(b), where the tag bit is appended to the end of the data word and encoded into the last TLC. Since 2^n ($n \geq 1$) can never be a multiple of 3, the tag bit can always be encoded within the existing TLCs without any overhead for word sizes that are integer powers of 2. Furthermore, to ensure that the tag bit does not impact latency, we reserve state '7' of the TLC to indicate CompEx-coded data.

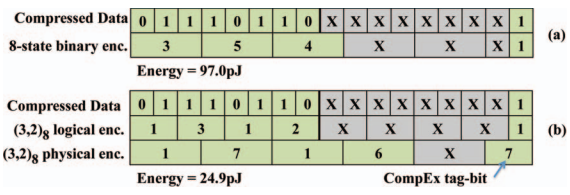

Figure 5: **Illustration of CompEx coding. The key take-away is that CompEx coding reduces energy for no overhead since the tag bit is encoded within the last TLC.**

CompEx coding and ECC: MLC/TLC NVMs are susceptible to both soft and hard errors, which necessitates the use of error detection and correction (EDAC) techniques such as ECC [52], ECP [53], etc. NVMs with EDAC usually use separable coding techniques, i.e., the data field is stored separate from the EDAC field. Since a write operation may alter both the data and EDAC fields, the benefits of CompEx coding the data field may be nullified by high latency writes in the EDAC field. In order to address this potential shortcoming, we propose that whenever the data field is compressible, the EDAC field be written in expansion-coded form — the additional cells required for expansion coding the EDAC field can be obtained from the residual cells after compression of the data field. From Tables 1 & 2, we can infer that FPC-based (BΔI-based) CompEx coding leaves 4 (30) TLCs unused in the worst case, which can be purposed to provide EDAC in expansion-coded form to preserve/extend the latency/energy benefits of CompEx coding. On the other hand, if the data field is incompressible, both the data and EDAC fields are written using conventional binary coding.

CompEx coding and encryption: NVM non-volatility has emerged as a serious data security concern [54–56] — the stored data continues to persist in the memory even after the system is powered down, exposing sensitive data to a malicious intruder. The dominant proposal to thwart such attacks advocates data encryption before writing to the NVM main memory. However, encryption scrambles the data and potentially reduces the data regularity that is necessary for CompEx coding. This limitation can be circumvented by integrating the CompEx codec with the encryption engine. By compressing data before encryption, it is possible to use expansion coding on the encrypted data to preserve most of the benefits of CompEx coding. Since the encryption engines are mostly located in the memory controller within the processor [55, 56], this proposal requires that the CompEx codec move from the NVM module to the processor. A comprehensive investigation of the interplay between CompEx coding and encryption is outside the scope of this paper.

3.3.1 Theoretical analysis of CompEx coding

As we have established, CompEx coding can be applied without additional memory overhead, we now theoretically estimate the energy reduction for TLC RRAM whose energy and latency numbers are given by Fig. 2. We start by calculating the average energy for TLCs with 8 states, followed by the average energy for only 4 lowest energy TLC states. First, assuming that the write data follows uniform random distribution [57], the average/expected energy for a TLC with 8 equally likely states, i.e., binary coding, is given by the sum of the entries for energy for all the states in Fig. 2 divided by 8 (= 16pJ for this case), i.e., $\mathrm{E}[\mathrm{Energy}_{\mathrm{binary}}] = \sum_{i=0}^{7} e_i = 16\mathrm{pJ}$. Here, e_i is the energy required for programming the TLC RRAM to i^{th} state. Second, the average energy for a TLC that utilizes only 0, 1, 6, and 7 states, i.e., the $(3,2)_8$ expansion coding, is given by the sum of all energy for these subset of states divided by 4 (= 4.7pJ for this case), i.e., $\mathrm{E}[\mathrm{Energy}_{(3,2)_8}] = \sum_{i=0,1,6,7} e_i = 4.7\mathrm{pJ}$. Since $(3,2)_8$ expansion code uses $(3/2)\times$ more TLCs over binary encoding, we also need to factor this number into our energy computation. Therefore, the overall reduction in energy by using $(3,2)_8$ expansion code for this example is $\mathrm{E}[\mathrm{Energy}_{\mathrm{binary}}]/\mathrm{E}[\mathrm{Energy}_{(3,2)_8}] = (16/4.7)\,(2/3) \approx \mathbf{2.3}$. Additionally, since a fraction of the data is left uncompressed by FPC, the overall reduction in energy by using CompEx coding in comparison to binary encoding is $\mathrm{E}[\mathrm{Energy}_{\mathrm{binary}}]/\mathrm{E}[\mathrm{Energy}_{\mathrm{CompEx}}] = \mathbf{2.3}\,p$, where p is the compression ratio of the compression scheme. Similarly, for $(6,5)_8$ expansion code, the average energy reduction can be derived to be $1.4\times$ using the same procedure (not discussed here for brevity). Our evaluation of expansion coding using real-world workloads (discussed in Section 5.1) closely agrees with the theoretical results, i.e, $2.2\times$ and $1.25\times$ practical energy reduction in comparison to $2.3\times$ and $1.4\times$ derived theoretically for $(3,2)_8$ and $(6,5)_8$ expansion coding, respectively.

3.3.2 CompEx codec architecture

This section describes the architecture that integrates CompEx coding within the NVM module controller. The architecture [3] for CompEx coding consists of the CompEx code encoder on the write-path and the CompEx code decoder on the read-path, as shown in Fig. 7. The CompEx codec (encoder-decoder) logic is embedded in the NVM module controller between the NVM array and the data bus to seamlessly encode and decode the accessed data as shown in Fig. 7. The CompEx code encoder is made up of compression logic followed by an expansion code encoder. Similarly, the CompEx code decoder is made up of a expansion code decoder followed by decompression logic. FPC-based CompEx coding uses word-size write/read accesses, while BΔI-based CompEx coding uses cache-line size accesses.

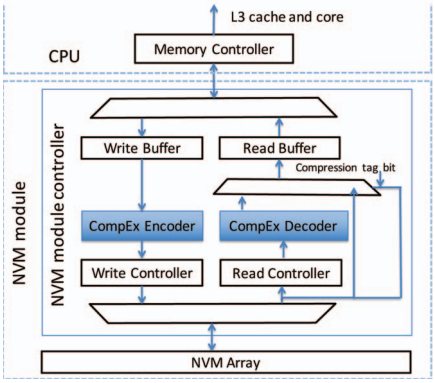

Figure 7: **CompEx codec architecture. The codec lies in the read/write data path adding 2/3 cycles overhead, respectively. However the latency reduction obtained due to CompEx coding far out weigh this overhead.**

Write: Whenever the MLC/TLC NVM module receives a write access from the memory controller on the processor side, the CompEx codec handles them as follows. First, the incoming data is passed through the compression logic, which compares the data with all the compression patterns to attempt data compression. If the data is compressible, then the compressed data is passed through the expansion code encoder; uncompressible data are directly sent to the write circuit. The expansion code encoder encodes every

[3]For a multi-channel memory system, each channel uses a dedicated codec, which results in a small overhead (Section 3.3.3) for the realized performance gains.

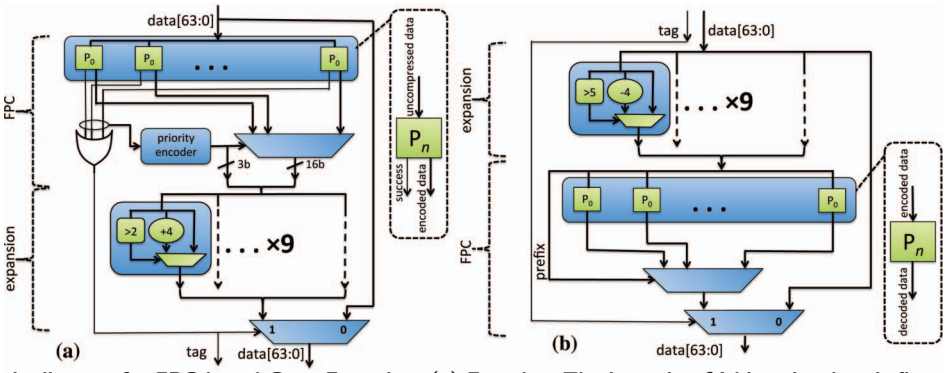

Figure 6: **Schematic diagram for FPC-based CompEx codec. (a) Encoder: The incoming 64-bit write-data is first encoded using the FPC encoder (latency = 2 cycles) followed by the expansion coding encoder (latency = 1 cycle) to obtain encoded data and tag bit, (b) Decoder: The tag bit and the encoded data are first decoded for expansion coding (latency = 1 cycle) and then decoded for FPC (latency = 1 cycle). Our implementation hides these latencies using memory access pipelining.**

2-bit slice of the compressed data to 3-bit codewords. However, since the width of the encoded data is always less than 64 bits (for FPC) and 64 bytes (for BΔI), CompEx coding has zero memory overhead. Note that the tag bit may need to be updated to record the outcome of CompEx coding, regardless of whether it is successful or unsuccessful. But, as discussed earlier, this tag bit can be concatenated and absorbed into the data using the last TLC without requiring an extra cell for the tag bit.

Read: When the MLC/TLC NVM module receives a read access from the memory controller, the CompEx codec inside the NVM module controller decodes the data before forwarding it to the memory controller. The read circuit reads the cells of the whole word and forwards the data to the CompEx codec. The tag bit is recovered from the last TLC of the read word and used to determine whether the data has to be CompEx decoded or forwarded directly to the NVM module controller.

3.3.3 CompEx codec logic overhead

To estimate the logic overhead, we designed and synthesized 64-bit FPC-based CompEx codec (Fig. 6) using Verilog. The design of BΔI-based compressor is assumed from the original BΔI proposal [39]. The estimated logic hardware overhead for the CompEx codec tabulated in Table 3 is ≈10k gates (< 0.1% per NVM module). Furthermore, although the CompEx codec has an estimated latency of 3/2 cycles for encoding/decoding, respectively, our implementation uses memory access pipelining to hide this in practice.

Direction	Logic	Instances
Read-path	64-bit 2-to-1 multiplexer	8
	3-bit subtractor	9
Write-path	16-bit comparator (equality check only)	3
	2-bit comparator	32
	1-bit 2-to-1 multiplexer	160
	3-bit adder	9

Table 3: **Logic overhead for synthesized CompEx codec.**

4. Methodology

Our evaluation of CompEx coding is based on (i) full-system simulations to evaluate the system-level performance of CompEx coding and (ii) trace-based simulations for deep, multi-billion instruction evaluation of memory-level energy and latency of CompEx coding.

4.1 Full-system simulation

CompEx coding is evaluated using full-system simulations of a system that integrates TLC RRAM memory using MARSS [11], a full-system multi-core simulator, and DRAMSim2 [12], a cycle-accurate main memory simulator.

MARSS uses x86 core models from PTLSim [58], a cycle-accurate x86 micro-architecture simulator and plugs it into QEMU [59], a binary-translation system for emulating full systems. QEMU provides the capability of emulating various I/O devices (HDD, ethernet, HID, etc.) that can be used to bootup entire operating systems without any modification (Linux in this work). Note that in this work, we modify MARSS to propagate write data along with the address throughout its memory hierarchy.

We use DRAMSim2, a cycle-accurate main memory simulator for simulating the DDR3 NVM main memory system. MARSS and DRAMSim2 are integrated to provide a monolithic, seamless, cycle-accurate simulation of the entire system. Since each access in a TLC NVM memory can potentially have different access latencies, we modify DRAMSim2 to account for this.

MARSS setup: MARSS was configured to simulate a standard 4-core out-of-order system running at 3GHz. Each core has its own L1 cache with 2 separate instances of 32kB SRAM for data and instructions; the L2 cache is private, with each core having its own instance of 256kB SRAM; finally, the L3 cache is a single, shared, write-back cache of size 8MB. The latencies of each level of cache is tabulated in Table 4.

Parameter	Attributes	Latency
Core	4 cores, dual issue out-of-order	—
L1 (instruction) cache	32kB, 2 way	2ns
L1 (data) cache	32kB, 2 way	2ns
L2 cache	256kB, private, 8 way	5ns
L3 cache	8MB, shared, 16 way	20ns
Cache line size	64 bytes	—
Main memory	16GB, 8 banks, 1 channel (DRAMSim2), 1GHz	160ns (average)

Table 4: **Configuration of the evaluation architecture.**

DRAMSim2 setup: For a fairly accurate timing simulation, we modified DDR timing parameters along the lines of [1] for substituting DRAM with NVM. We use TLC RRAM with latency parameters extracted and summarized in [9].

Workloads: We evaluate CompEx coding using SPEC CPU-2006 [40] benchmark suite. These benchmarks reflect a va-

riety of integer and floating-point based workloads used by modern computing systems. To evaluate real-world usage, we use 9 composite workloads with each workload containing 4 benchmarks from the SPEC CPU2006 benchmark suite. These composite workloads are derived from [60], where benchmarks are selectively picked due to their memory intensive nature. Table 5 lists the constituent benchmarks for each composite workload and their corresponding writes per kilo-instruction (WPKI) and misses per kilo-instruction (MPKI) as reported by MARSS. For a fair study of memory system (preventing benchmarks from competing among themselves), each benchmark in a workload is tied to a core such that it can execute only on that core.

Workload	Constituent benchmarks	WPKI	MPKI
WD_1	leslie3d, leslie3d, mcf, mcf	4.61	9.66
WD_2	lbm, leslie3d, libquantum, mcf	6.01	12.55
WD_3	lbm, lbm, libquantum, libquantum	8.60	17.57
WD_4	bwaves, leslie3d, omentpp, sphinx3	3.68	7.95
WD_5	GemsFDTD, libquantum, milc, zeusmp	3.72	7.93
WD_6	GemsFDTD, libquantum, milc, milc	5.00	10.44
WD_7	bzip, libquantum, milc, omentpp	7.41	15.32
WD_8	cactusAMD, gcc, gobmk, zeusmp	4.48	10.07
WD_9	astar, gobmk, hmmer, soplex	5.01	10.80

Table 5: **Composite workloads comprising of 4 benchmarks from SPEC CPU2006 benchmark suite (derived from [60]) for full-system evaluation of CompEx coding. The WPKI and MPKI numbers are sensitive to the CPU and memory system architecture; the numbers presented in this table are compiled for the architecture defined in Table 4 using MARSS.**

4.2 Trace-driven simulation

For running deep, multi-billion-instruction simulations, CompEx coding is evaluated using both an in-house trace-driven simulator for evaluating the memory array level dynamic energy and NVMain [41] — an architectural-level main memory simulator for emerging non-volatile memories — for evaluating the total energy at the memory module level. We modified NVMain to reflect the variable-write-latency behavior of CompEx coding and also configured NVMain to simulate an architecture equivalent to that in Table 4. The traces are generated from the SPEC CPU2006 [40] benchmark suite using Intel Pin binary instrumentation tool [42] on a machine running a 3.3 GHz Intel Core i7 CPU. Note that we also use Gem5 system simulator [61] to validate these results using an equivalent architecture; the results are consistent with what is reported here and not discussed for brevity. Our simulation framework captures memory accesses from the processor, recording only those accesses sent to main memory. During trace generation, the benchmarks are first run through 5×10^5 memory writes, to ignore the write accesses from program initialization; they are then run until 4×10^6 memory write operations (equivalent to about 4 billion instructions on average) have been recorded or until the program terminates.

5. Evaluation and results

This section presents the evaluation of CompEx coding at the memory and system levels. First, we present the energy and latency results at the memory level for different encodings — baseline (binary encoding with DCW), compression techniques (FPC and BΔI), expansion coding ($(3,2)_8$ and $(6,5)_8$), and CompEx coding (FPC-based and BΔI-based CompEx coding, using $(3,2)_8$ expansion coding). Second, we present the results for system-level evaluation of CompEx coding; primarily to evaluate the impact of latency improvements of CompEx coding on system performance.

5.1 Memory energy/latency

Summary: Table 6 summarizes and compares the total module energy, memory array dynamic energy, latency, and overhead for the 7 encoding techniques considered in this paper. In summary, FPC-based and BΔI-based CompEx coding reduces the memory array write energy by 31% and 74%, write latency by 28% and 21%, respectively, in comparison to binary encoding for no overhead.

	Total Energy	Dyn. Energy	Dyn. Latency	Ovhd.
Binary coding	100%	100%	100%	0%
[26] extended to 64-bit FPC	95%	87.6%	96.6%	0%
BΔI compression	58%	35.2%	99.4%	0%
$(3,2)_8$ expansion coding	31%	46.3%	39.5%	50%
$(6,5)_8$ expansion coding	83%	80.2%	72.8 %	20%
FPC-based CompEx coding	76%	69.5%	72.2%	0%
BΔI-based CompEx coding	47%	25.9%	80.2%	0%

Table 6: **Comparison of memory array write energy, latency, and overhead normalized to classical binary coding for the 7 schemes considered in this paper. Note that all the cases use classical read-modify-write (DCW) [14] to update only the modified cells in the NVM array.**

Energy: Our simulation framework tracks all the cell writes that occur from the beginning of program execution to compute the cumulative energy required. It is important to note that although CompEx coding does not require any additional memory overhead in comparison to classical binary encoding, the static energy from peripheral circuits and the memory array are indirectly influenced by the reduction in the latency of each write operation. A lower write latency translates to a lower energy expense to keep the peripheral circuits active, which is evaluated using NVMain [41]. Figures 8(a) and 8(b) show the memory array dynamic energy and the total memory module energy, respectively, for FPC, BΔI, $(3,2)_8$ and $(6,5)_8$ expansion coding, and FPC-based and BΔI-based CompEx coding, normalized to binary coding. The last entries of Fig. 8 represent the geometric mean of the energy reduction across all the benchmarks, which is equivalent to simulating all these benchmarks for the same execution time. Note that all the cases use classical read-modify-write (DCW) [14] for writing only the modified cells to the NVM array. Simulations show that FPC-based CompEx coding reduces the memory array dynamic energy (total memory module energy) by 31% (24%) and 21% (20%) in comparison to binary coding and FPC, respectively, while BΔI-based CompEx coding reduces energy by 74% (53%) and 21% (19%) in comparison to binary coding and BΔI, respectively. Additionally $(3,2)_8$ and $(6,5)_8$ expansion coding in isolation show a reduction of $2.2\times$ and $1.25\times$ in memory array write energy in comparison to binary coding, which is in close agreement with the theoretical estimate of $2.3\times$ and $1.4\times$, respectively, as derived in Section 3.3.1.

Latency: We evaluate the impact on memory array write latency due to CompEx coding. As detailed in Section 2, the states with lower energy also require lower write latencies due to the iterative P&V procedure. Whereas the energy re-

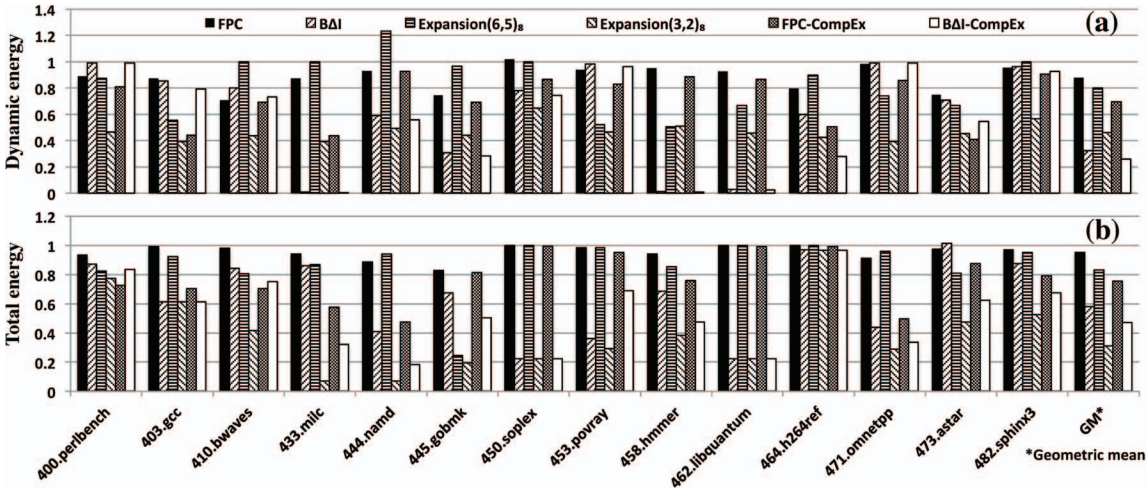

Figure 8: **(a) Dynamic energy at the memory array level and (b) total memory module energy results (using NVMain [41]) for FPC, $(3,2)_8$ expansion coding, $(6,5)_8$ expansion coding, FPC-based CompEx coding, and BΔI-based CompEx coding, normalized to classical binary coding. FPC-based and BΔI-based CompEx coding reduce dynamic energy (total energy) by 31% (24%) and 74% (53%) in comparison to binary encoding. As expected (in Table 2), the dynamic energy performance of BΔI-based CompEx coding is better than FPC-based CompEx coding.**

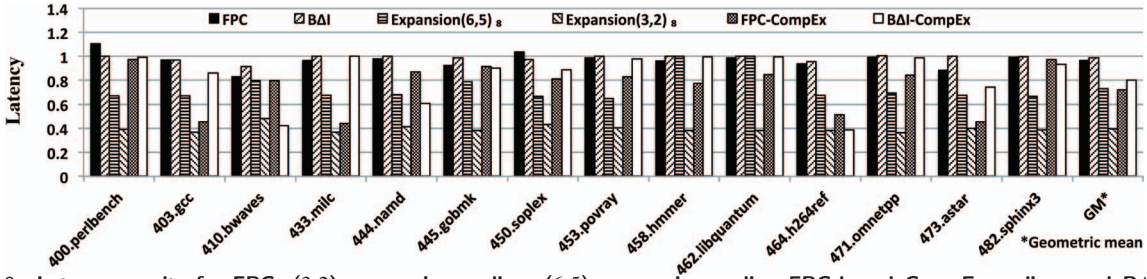

Figure 9: **Latency results for FPC, $(3,2)_8$ expansion coding, $(6,5)_8$ expansion coding FPC-based CompEx coding, and BΔI-based CompEx coding, normalized to classical binary coding. FPC-based and BΔI-based CompEx coding reduce latency by 28% and 20% in comparison to binary encoding. As expected (in Table 2), the latency performance of FPC-based CompEx coding and BΔI-based CompEx coding are comparable to each other.**

quired for writing a word is given by the sum of the energy required for all the cells, the latency for writing a word depends on the cell that requires the longest latency. Therefore, since CompEx coding encodes data using only lower energy states, the program latency for writing compressed data is also reduced. Our simulation determines the latency for each write access individually by tracking the maximum latency cell-write for the word access. The write latency for each access is then cumulatively computed to obtain the overall latency for program execution. Fig. 9 shows that FPC-based CompEx coding is able to reduce the overall write latency by 28% and 17% in comparison to binary coding and FPC, respectively, and BΔI-based CompEx coding reduces overall write latency by about 65% and 64% in comparison to binary coding and BΔI, respectively.

5.2 Full-system evaluation

In this section, we report and discuss the impact of CompEx coding on system performance. We use two metrics: (i) instructions-per-cycle (IPC), which is a metric for full-system performance and (ii) main memory bandwidth, which is a metric for main memory performance.

Summary: Table 7 summarizes and compares IPC and bandwidth for classical binary encoding, FPC-based CompEx coding, and BΔI-based CompEx coding. In summary, FPC-based and BΔI-based CompEx coding improves IPC by 6.3% and 5.1%, and memory bandwidth by 11.1% and 12.6% in comparison to binary encoding, respectively.

	IPC	Bandwidth
Binary coding	100%	100%
FPC-based CompEx coding	106.3%	111.1%
BΔI-based CompEx coding	105.1%	112.6%

Table 7: **IPC and bandwidth for FPC-based CompEx coding and BΔI-based CompEx coding, normalized to classical binary encoding. Note that all the cases use DCW [14] to update only the modified cells in the NVM array.**

Instructions per cycle (IPC): To evaluate the impact of CompEx coding on IPC, we use a full-system simulator based on MARSS [11] and DRAMSim2 [12]. The simulation setup is described in detail in Section 4. We simulate nine composite workloads from Table 5. Figure 10 shows the IPC for each benchmark in each workload. The last set of bars in each workload in Fig. 10 represents the mean IPC for that workload (since each benchmark is run on a separate and exclusive core, the mean IPC is given by the arithmetic mean of the IPCs of the constituent benchmarks of the workload [62, Ch. 1]). The last set of bars in Fig. 10 represents the harmonic mean of the IPCs of all the workloads (computing the harmonic mean is equivalent to running all the workloads in the same system for a fixed number of instructions). First, our simulations show a good correlation between IPC and both WPKI and MPKI. For example, consider the workloads WD_3 (highest MPKI) and WD_5 (lowest MPKI) — as expected, the high cache miss-rate of WD_3 lowers the IPC in comparison to WD_5 that has a lower cache miss-rate. To

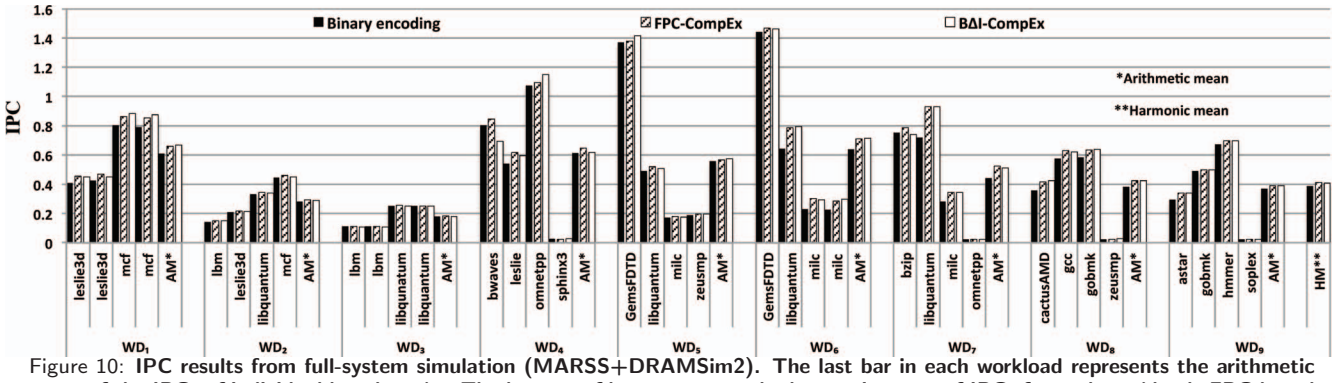

Figure 10: **IPC results from full-system simulation (MARSS+DRAMSim2). The last bar in each workload represents the arithmetic mean of the IPCs of individual benchmarks. The last set of bars represent the harmonic mean of IPCs for each workload. FPC-based CompEx coding and BΔI-based CompEx coding show an IPC improvement of 6.3% and 5.1% in over binary encoding with DCW.**

understand the dependence of IPC on WPKI, let us consider the workloads WD_7 (highest WPKI) and WD_4 (lowest WPKI). CompEx coding reduces latency and improves IPC only during write accesses. Thus the workload with higher WPKI should show a higher improvement in IPC in comparison to the workload with lower WPKI; our simulations, in agreement with the above argument, show that FPC (BΔI)-based CompEx coding improves the IPC by 17.5% (14.8%) for WD_7 (high WPKI) in comparison to 5.5% (1%) for WD_4 (low WPKI). Second, our simulations also show similar correlations at the individual benchmark level. Intuitively, memory-intensive benchmarks like milc (WD_6) or lbm (WD_3) should have lower IPC in comparison to less memory-intensive benchmarks like gemsFDTD (WD_6) or omnetpp (WD_4); our simulations are in excellent agreement with the above argument. Finally, our simulations show that using FPC-based and BΔI-based CompEx coding, the overall IPC improvement is about 6.3% and 5.1%, respectively, in comparison to classical binary encoding.[4]

Memory bandwidth: Fig. 11 shows the main memory bandwidth across nine composite workloads for the baseline and CompEx coding in comparison to classical binary encoding. In Fig. 11, the bars represent FPC-based CompEx coding and BΔI based CompEx coding normalized to the baseline (binary encoding with DCW). The last set of bars in Fig. 11 represents the geometric mean of the normalized improvements (computing geometric mean is equivalent to running each workload for the same execution time). Intuitively, similar to IPC, workloads that have high WPKI should contribute to higher bandwidth improvements in comparison to those with low WPKI. This can be seen using the example of workloads WD_6/WD_7 (high WPKI), which show an improvement of 30% in bandwidth using BΔI-based CompEx coding in comparison to WD_5 (low WPKI), which shows an improvement of only 6%. On the whole, our simulations show that FPC-based and BΔI-based CompEx coding improve the average memory bandwidth by 11.1% and 12.59%, respectively, in comparison to binary encoding.

[4]We see a good correlation between the latency results from trace-based simulations and the IPC results from full-system simulation. Benchmarks like milc and libquantum that have better improvement in latency using FPC-based CompEx coding in comparison to BΔI-based CompEx coding (from trace-based simulations) have also shown better IPC numbers for FPC-based CompEx coding in comparison to BΔI-based CompEx coding. On the other hand, benchmarks like sphinx3 and gobmk show that BΔI-based Compex coding provides better improvement in latency and IPC in comparison to FPC-based CompEx coding.

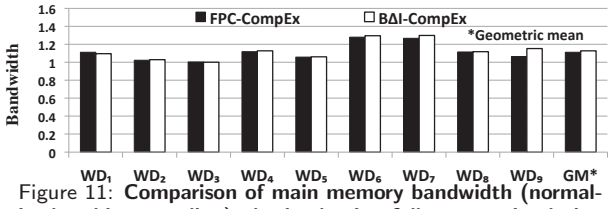

Figure 11: **Comparison of main memory bandwidth (normalized to binary coding) obtained using full-system simulation (MARSS+DRAMSim2). The overall bandwidth improvement is 11.1% and 12.6% for FPC-based CompEx coding and BΔI-based CompEx coding, respectively.**

Lifetime: In this paper, we theoretically evaluate the lifetime gains of CompEx coding using TLC RRAM as an example. [63] discusses the three primary mechanisms for cell failure in RRAM in detail and each mechanism contributes to limit the lifetime of a cell; at the memory array level, the lifetime of an RRAM cell is specified as a limit on the number of programming cycles (SET/RESET) it can endure before the cell becomes dysfunctional [63]. Therefore, the lifetime of MLC/TLC RRAM is lower in comparison to SLC RRAM since programming an MLC/TLC RRAM requires higher number of P&V cycles in comparison to SLC RRAM [7]. Since CompEx coding effectively reduces the number of P&V cycles for programming an NVM cell (by limiting cell states to only low energy/latency states), it also improves the lifetime of the MLC/TLC NVM. Lifetime evaluation of CompEx coding for TLC RRAM — along the lines of [7, 8, 64] — show that CompEx coding increases the lifetime by 1.8× over classical binary coding.

6. Related Work

MLC/TLC NVMs definitely benefit from the broad set of solutions developed to improve energy, latency, and lifetime of SLC NVMs [65]. However, SLC-based solutions do not address the energy/latency problems of MLC/TLC NVMs that is primarily due to the iterative nature of the cell-write operation, i.e., iterative P&V. Solutions that explicitly address the challenges of working with MLC/TLC NVMs have focused on data compression and data coding by exploiting the physical properties of the NVM cell and the locality in data traffic [30, 37, 38].

Memory compression: Increasing cache capacity by various data compression techniques that leverage different localities have been studied [27–31]. On similar lines, there are solutions that compress main memory traffic for the benefits of bandwidth, power, and capacity [26, 32–34]. In the context of MLC/TLC NVMs, compressing main memory

traffic yields fewer cell-writes per write access, thus lowering energy. Whereas there are solutions that reduce MLC/TLC energy and latency by excluding undesirable states for additional memory area (summarized below), CompEx coding is to the best of our knowledge the first work to explicitly investigate the tradeoffs between the area recovered from compression and solutions that exclude undesirable MLC/TLC states.

Excluding undesirable states: [66, 67] propose circuit and architectural changes to dynamically configure MLC PCM cells as either MLC or SLC for latency benefits. On the other hand, [35] excludes programming the hard-to-reset cells that require high programming current and recovers the lost data using ECC, thereby reducing the power consumption and improving lifetime. Extending the observations of [35], Elastic RESET (ER) [8] proposes data coding that eliminates the undesirable terminal RESET state (high programming current) and uses only 2 or 3 of the 4 states of a cell to realize lifetime/power/latency improvements. Similar to ER, hybrid MLC/SLC [36] proposes to opportunistically use PCM cells as either MLC/SLC for energy and latency benefits. Independently, for MLC PCM, [37] proposes energy-efficient data coding to reduce the usage of intermediate high-energy states by mapping the most frequent data patterns to the low energy states. However, [37] requires online computation and storage of the most frequent patterns for every memory line at runtime, incurring compute, memory, and logic overhead. For MLC PCM, [5] proposes data coding that eliminates one of the intermediate resistance states; this improves cell retention but incurs memory overhead. For TLC RRAM, [9] proposes data coding that uses 6 out of 8 TLC RRAM states to improve latency and energy by eliminating the use of intermediate resistance states [9, 37]. By combining IDM with dynamic data remapping and error-correcting pointers, lifetime improvements for 20% memory overhead and negligible impact on energy/latency are reported. Recently, [68] observed a super-linear relationship between RESET latency and the number of 1s written to the array, and advocated writing smaller chunks of data using compression to reduce the RESET latency. For MLC PCM, form-switch (FS) [24] first introduced the notion of writing data in SLC/MLC depending on the result of compression. However, FS may not always result in energy/latency reduction since it depends upon the compression technique used and the energy/latency profile of the NVM cell. In practice, it is necessary to balance dynamic tradeoffs between data compression, the NVM energy/latency profile, and data encoding: CompEx coding as proposed in this paper is a step in this direction to realize simultaneous improvements in energy, latency, and lifetime of MLC/TLC NVMs.

7. Conclusions

This paper described CompEx coding, a low overhead, dynamic tradeoff framework that synergistically integrates statistical compression with expansion coding to realize simultaneous energy, latency, and lifetime improvements in MLC/TLC NVMs. The core idea of CompEx coding is to ***selectively apply expansion codes***, i.e., linear block codes that encode data using only the low energy states of an MLC/TLC cell ***to compressed data***, thereby ensuring that the resulting data in expansion-coded form will not exceed the original data width. CompEx coding is agnostic to the choice of compression technique; in this paper, we evaluated CompEx coding using both frequent pattern compression (FPC) and base-delta-immediate (BΔI) compression. CompEx coding integrates FPC/BΔI with $(k,m)_q$ 'expansion' coding; expansion codes are a class of q-ary linear block codes that encode data using only the low energy states of a q-ary NVM cell. Our full-system simulations of a system that integrates TLC RRAM show that CompEx coding reduces total memory energy by 53% and cell latency by 24%; these improvements translate to a 5.7% improvement in IPC, a 11.8% improvement in main memory bandwidth, and 1.8$\times$ improvement in lifetime over classical binary coding using data-comparison write. CompEx coding thus addresses the programming energy/latency as well as the lifetime challenges of MLC/TLC NVMs that pose a serious technological roadblock to their adoption in high performance computing systems.

References

[1] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[2] I. Baek, M. Lee, S. Seo, M.-J. Lee, D. Seo, D.-S. Suh, J. Park, S. Park, T. Kim, I. Yoo, U.-i. Chung, and J. Moon, "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proc. Intl. Electron Devices Meeting*, 2004.

[3] "International technology roadmap for semiconductors," 2011.

[4] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. C. Buda, F. Pellizzer, D. W. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE Journal of Solid-state Circuits*, vol. 44, no. 1, 2009.

[5] D. H. Yoon, J. Chang, R. Schreiber, and N. Jouppi, "Practical nonvolatile multilevel-cell phase change memory," in *Proc. Intl. Conference for High Performance Computing, Networking, Storage and Analysis*, 2013.

[6] Nirschl, J. Philipp, T. Happ, G. Burr, B. Rajendran, M. Lee, A. Schrott, M. Yang, M. Breitwisch, C. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y. Chen, Y. Zhu, R. Bergmann, H. Lung, and C. Lam, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *IEEE Intl. Electron Devices Meeting*, 2007.

[7] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. Design Automation Conference*, 2013.

[8] L. Jiang, Y. Zhang, and J. Yang, "ER: Elastic RESET for low power and long endurance MLC," in *Proc. Intl. Symposium on Low Power Electronics and Design*, 2012.

[9] D. Niu, Q. Zou, C. Xu, and Y. Xie, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *Proc. Intl. Conference on Computer Design*, 2013.

[10] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, 1995.

[11] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: a full system simulator for multicore x86 CPUs," in *Proc. Design Automation Conference*, 2011.

[12] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, 2011.

[13] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[14] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu, "A low power phase change random access memory using a data-comparison write scheme," in *Proc. Intl. Symposium on Circuits and Systems*, 2010.

[15] P. Palangappa, J. Li, and K. Mohanram, "WOM-Code solutions for low latency and high endurance in phase change memory," *IEEE Transactions on Computers*, 2015.

[16] P. M. Palangappa and K. Mohanram, "Flip-Mirror-Rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories," in *Proc. Great Lakes Symposium on VLSI*, 2015.

[17] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2010.

[18] M. Qureshi, M. Franceschini, A. Jagmohan, and L. Lastras, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write time," in *Proc. Intl. Symposium on Computer Architecture*, 2012.

[19] Y. Kim, S. Yoo, and S. Lee, "Write performance improvement by hiding R drift latency in phase change RAM," in *Proc. Design Automation Conference*, 2012.

[20] S. Kwon, S. Yoo, S. Lee, and J. Park, "Optimizing video application design for phase-change RAM-based main memory," *IEEE Transactions VLSI Systems*, vol. 20, no. 11, 2012.

[21] M. K. Qureshi, J. Karidis, M. Fraceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of phase change memories via start-gap wear leveling," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[22] G. Wu, H. Zhang, Y. Dong, and J. Hu, "CAR: Securing PCM main memory system with cache address remapping," in *Proc. Intl. Conference on Parallel and Distributed Systems*, 2012.

[23] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proc. Intl. Symposium on Computer Architecture*, 2010.

[24] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. Childers, "Improving write operations in MLC phase change memory," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2012.

[25] J. Yue and Y. Zhu, "Making write less blocking for read accesses in phase change memory," in *Proc. Intl. Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, 2012.

[26] D. Dgien, P. Palangappa, N. Hunter, J. Li, and K. Mohanram, "Compression architecture for bit-write reduction in non-volatile memory technologies," in *Proc. Intl. Symposium Nanoscale Architectures*, 2014.

[27] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching," in *Proc. Intl. Symposium on Microarchitecture*, 2013.

[28] E. Ahn, S.-M. Yoo, and S.-M. S. Kang, "Effective algorithms for cache-level compression," in *Proc. Great Lakes Symposium on VLSI*, 2001.

[29] X. Chen, L. Yang, R. Dick, L. Shang, and H. Lekatsas, "C-Pack: A high-performance microprocessor cache compression algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2010.

[30] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *Proc. Intl. Symposium on Microarchitecture*, 2000.

[31] J. Yang and R. Gupta, "Energy efficient frequent value data cache design," in *Proc. Intl. Symposium on Microarchitecture*, 2002.

[32] N. R. Mahapatra, J. Liu, K. Sundaresan, S. Dangeti, and B. V. Venkatrao, "A limit study on the potential of compression for improving memory system performance, power consumption, and cost," *Journal of Instruction-Level Parallelism*, 2005.

[33] K. S. Yim, J. Kim, and K. Koh, "Performance analysis of on-chip cache and main memory compression systems for high-end parallel computers," in *Proc. Intl. Conference on Parallel and Distributed Processing Techniques and Applications*, 2004.

[34] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *Proc. Intl. Symposium on Computer Architecture*, 2005.

[35] L. Jiang, Y. Zhang, B. Childers, and J. Yang, "FPB: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proc. Intl. Symposium on Microarchitecture*, 2012.

[36] H. G. Lee, S. Baek, J. Kim, and C. Nicopoulos, "A compression-based hybrid MLC/SLC management technique for phase-change memory systems," in *Proc. IEEE Annual Symposium on VLSI*, 2012.

[37] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. Intl. Conference on Computer Design*, 2011.

[38] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches," tech. rep., University of Wisconsin-Madison, 2004.

[39] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. Intl. Conference on Parallel architectures and compilation techniques*, 2012.

[40] "SPEC CPU2006," 2006.

[41] M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 392–397, 2012.

[42] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. Conference on Programming Language Design and Implementation*, 2005.

[43] K.-J. Lee, B.-H. Cho, W.-Y. Cho, S. Kang, B.-G. Choi, H.-R. Oh, C.-S. Lee, H.-J. Kim, J. min Park, Q. Wang, M.-H. Park, Y.-H. Ro, J.-Y. Choi, K.-S. Kim, Y.-R. Kim, I.-C. Shin, K. won Lim, H.-K. Cho, C.-H. Choi, W. ryul Chung, D.-E. Kim, K.-S. Yu, G.-T. Jeong, H.-S. Jeong, C.-K. Kwak, C.-H. Kim, and K. Kim, "A 90 nm 1.8 v 512 Mb diode-switch PRAM with 266 MB/s read throughput," *IEEE Journal of Solid-state Circuits*, vol. 43, no. 1, 2008.

[44] M. Kang, T. Park, Y. Kwon, D. Ahn, Y. Kang, H. Jeong, S. Ahn, Y. Song, B. Kim, S. Nam, H. Kang, G. Jeong, and C. Chung, "PRAM cell technology and characterization in 20nm node size," in *Intl. Electron Devices Meeting*, 2011.

[45] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. Chen, and M.-J. Tsai, "Metal oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, 2012.

[46] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. Intl. Symposium on High-performance Computer Architecture*, 2013.

[47] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proc. of Intl. Symposium on High-Performance Computer Architecture*, 2012.

[48] S. Kang, W. Y. Cho, B.-H. Cho, K.-J. Lee, C.-S. Lee, H.-R. Oh, B.-G. Choi, Q. Wang, H.-J. Kim, M.-H. Park, Y. H. Ro, S. Kim, C.-D. Ha, K.-S. Kim, Y.-R. Kim, D.-E. Kim, C.-K. Kwak, H.-G. Byun, G. Jeong, H. Jeong, K. Kim, and Y. Shin, "A 0.1-μm 1.8-V 256-Mb phase-change random access memory (PRAM) with 66-mhz sysnchronous burst-read operation," *IEEE Journal of Solid-state Circuits*, vol. 42, no. 1, 2007.

[49] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8V 8GB PRAM with 40MB/s program bandwidth," in *Proc. Intl. Solid-state Circuits Conference*, 2012.

[50] A. Jog, A. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. Das, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. Design Automation Conference*, 2012.

[51] D. Costello and S. Lin, *Error control coding*. Pearson Higher Education, 2004.

[52] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *Proc. Intl. Symposium on High Performance Computer Architecture*, IEEE, 2012.

[53] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. Intl. Symposium on Computer Architecture*, 2010.

[54] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *Proc. Intl. Symposium on Computer Architecture*, 2011.

[55] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[56] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," in *Intl. Conference on Dependable Systems and Networks*, 2010.

[57] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[58] M. T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *IEEE Intl. Symposium on Performance Analysis of Systems and Software*, 2007.

[59] F. Bellard, "QEMU, a fast and portable dynamic translator.," in *USENIX Annual Technical Conference*, 2005.

[60] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lee, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2013.

[61] N. Bikerst, "The gem5 simulator," *IEEE Computer Architecture Letters*, vol. 39, no. 2, 2011.

[62] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach (5th Edition)*. Morgan Kaufmann Press, 2011.

[63] B. Chen, Y. Lu, B. Gao, Y. Fu, F. Zhang, P. Huang, Y. Chen, L. Liu, X. Liu, J. Kang, *et al.*, "Physical mechanisms of endurance degradation in TMO-RRAM," in *Intl. Electron Devices Meeting*, 2011.

[64] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *ACM SIGARCH Computer Architecture News*, 2010.

[65] M. Zhao, L. Jiang, Y. Zhang, and C. J. Xue, "SLC-enabled wear leveling for MLC PCM considering process variation," in *Proc. Design Automation Conference*, 2014.

[66] M. Arjomand, A. Jadidi, A. Shafiee, and H. Sarbazi-Azad, "A morphable phase change memory architecture considering frequent zero values," in *Proc. Intl. Conference on Computer Design*, 2011.

[67] X. Dong and Y. Xie, "AdaMS: Adaptive MLC/SLC phase-change memory design for file storage," in *Proc. Asia and South Pacific Design Automation Conference*, 2011.

[68] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2015.