A Locality-Aware Compression Scheme for Highly Reliable Embedded Systems

Juhyung Hong, Jeongbin Kim, Sangwoo Han, and Eui-Young Chung, *Member, IEEE*

***Abstract*—Dynamic random access memory (DRAM) reliability has become one of the critical issues in embedded systems, as DRAM process technology advances with the increase in bit error probability. Unfortunately, redundant error-correction code (ECC) chips cannot be applied to embedded systems since cores and DRAMs are tightly coupled without a dual in-line memory module (DIMM) slot to account for the form factor, cost, and limited pin count. Therefore, ECC parities are typically placed in the same physical array where the user and system data reside. This coexistence eventually deteriorates data locality, which could be the critical factor in DRAM performance degradation. To address this issue, we propose an ECC scheme called locality-aware compression (LoComp) which integrates a compression algorithm, DRAM data layout, and memory controller especially optimized for embedded systems. We focus on the locality of the dataset and its corresponding metadata, as well as spatial data locality in the design of DRAM data layout, which reduces the number of row activations. The major feature in a compression algorithm is adjusting the misalignment of data streams caused by the data packing in many embedded systems. Moreover, we specialize the memory controller to reduce DRAM access for ECC parities and compression flags. The core technologies for the memory controller are the adoption of a set of small caches for metadata and the support of partial write operation without changing the DRAM interface. LoComp+, an enhanced version of LoComp, further reduces DRAM access for metadata by placing the metadata close to the corresponding data. In the experiment, previous works increase the DRAM access time from 68% to over twice the value compared to ECC DIMM. Whereas, LoComp and LoComp+ show reduced performance degradation by 33% and 48%, respectively. In other words, LoComp and LoComp+ substantially improved performance from between 13% and 33% compared to previous embedded ECC schemes.**

***Index Terms*—Compression, dynamic random access memory (DRAM), embedded systems, error-correction code (ECC), locality, reliability, single error correction double error detection (SECDED).**

I. Introduction

SCALING of dynamic random access memory (DRAM) process technology continuously reduces the DRAM cost-per-bit. However, concerns have been raised over DRAM reliability stemming from deep submicron effects, such as the limited number of electrons, process variation, and coupling effects between intercells [1]. Such negative impacts on reliability are among the critical bottlenecks to the further scaling of DRAM technology. To address this concern, server and/or datacenter systems are equipped with error-correction code dual in-line memory modules (ECC DIMMs) or a high-level reliability technique called Chipkill [2]. These technologies isolate ECC parities (ECCs) from the data by placing the bits into a separate DRAM chip; hence, data locality is not affected by ECCs. In other words, system performance is not degraded by adopting these technologies from the locality perspective.

The reliability issue of DRAMs is also critical in embedded systems and many have adopted ECC schemes. Unfortunately, aforementioned ECC techniques are not applicable to embedded systems because of the form factor issue, costs, limited pin counts, and so on. These limitations force system designers to place ECCs and data within the same physical DRAM space, which leads to two side effects.

First, the processor in the embedded system requires additional bandwidth and latency for ECC DRAMs. In server systems, ECCs are stored in a separate DRAM chip; hence the data and its ECC are fetched in parallel manner. In contrast, embedded systems serially fetch them since they are located in the same physical DRAM, which results in additional bandwidth consumption per chip or performance degradation.

Second, the mix of data and corresponding ECCs in the same physical DRAM changes the physical address distance between the data, which strongly impacts data locality in the row buffer of DRAM. DRAM addressing is performed in multiple phases [3]. The first phase requires the long latency of precharge operation and row activation to move the entire row data into the row buffer. In the second phase, the row buffer serves data access after column access latency. Therefore, the change of data locality can increase latency for the first phase, and eventually degrade memory access performance.

Although these side effects are nonmarginal, many embedded systems are employing ECC schemes to address concerns over DRAM reliability. For instance, solid-state drives (SSDs) use parity or ECC for mapping tables in DRAM despite the bandwidth and performance degradation [4], since data center systems require end-to-end data integrity, which encompasses both the nonvolatile memory and internal buffer memory [5]. Other interesting embedded ECC schemes that maintain DRAM reliability without redundant ECC chips are introduced in [6]–[9].

Manuscript received June 1, 2017; revised August 14, 2017 and January 12, 2018; accepted March 5, 2018. Date of publication March 23, 2018; date of current version February 18, 2019. This work was supported in part by the National Research Foundation of Korea under Grant 2016R1A2B4011799, in part by the Ministry of Trade, Industry and Energy under Grant 10080722 and the Korea Semiconductor Research Consortium Support Program for the Development of the Future Semiconductor Device, and in part by the Samsung Electronics Company Ltd., Hwasung, Korea. This paper was recommended by Associate Editor J. Henkel. *(Corresponding author: Eui-Young Chung.)*

The authors are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea (e-mail: eychung@yonsei.ac.kr).

Digital Object Identifier 10.1109/TCAD.2018.2818692

One promising approach to tackle the abovementioned side effects is compression. Simply speaking, compression reduces the size of data to be fetched from DRAMs and largely reduces the first side effect, which is the increased bandwidth by ECCs. The storage overhead due to ECCs can be also reduced. Many studies have proposed compression techniques for cache [10], [11], main memory [12]–[14], SSD [15], [16], and GPU [17] to improve performance and/or storage utilization. To identify compression of data, it is necessary to have a compression flag (CF), which increases the complexity of data fetched from the DRAMs. The compression ratio can vary for each compressible data, which also results in complex address computation. To reduce this complexity, a fixed-size data layout has been used in many compressors. This method is advantageous when the compression ratio is higher than a certain threshold. In this case, the target data is compressed enough to fit into the fixed size with the ECC and the processor can fetch a cache line with a single burst like normal data.

On the other hand, if the compression ratio of the target data is lower than the threshold, the compressed data and ECC cannot fit into the fixed size format. In this case, the leftovers[1] must be stored somewhere else. Moreover, to track the leftover, the CF should keep the information on the leftover location. It is inevitable that the memory will be accessed multiple times to fetch the leftovers. An even worse case is when the fit-in data, leftovers, and CF are placed in different DRAM rows. In this case, the processor needs to open the row more than once, which results in expensive timing overhead compared to when these are placed in the same row. However, placing these three components in the same row could worsen the spatial locality of the DRAM row buffer if the given workload has a sequential property, since part of sequential data is expelled to a different location because of limited row size. Therefore, the data layout in compression schemes critically determines the overall performance, which is a challenging task.

Another issue in compression schemes is the placement of the compression engine. It is natural to place the compression engine with a last-level cache (LLC) at the early stage, since the LLC is the entry point to the main memory from the processor core. However, this approach increases the system bus traffic caused by the leftovers when the compression rate is lower than the threshold. Also, the storage controller in several embedded systems often accesses the main memory directly for paging. A good example of these systems is SSDs, where the page mapping table is frequently swapped in and out between the DRAM buffer and NAND flash memories. This trend directs the compression engine to be placed at the memory controller.

The last two issues to be discussed involve the access efficiency of metadata (e.g., CF and ECC). First, it is obvious that the CF should always be read before accessing the data in DRAM to check whether the data is compressed or not. Such overhead can be efficiently managed by adopting a flag cache for storing frequently accessed CFs [12], since the CFs of multiple cache lines can be accessed together with a single DRAM burst. The impact of the flag cache is highly dependent on its hit ratio.

[1]It can be data or ECCs depending the design of the data layout.

Second, the read–modify–write operation for modifying the metadata strongly impacts metadata access efficiency [9]. Since the length of the metadata for a cache line is usually shorter than a single DRAM burst, two DRAM bursts are required to modify the metadata. Such overhead can be greatly reduced by a partial write operation.

In short, the data layout of a compression scheme is very important in the performance of embedded systems supporting ECCs. Improvement of the compression ratio and data access efficiency is another crucial factor in the performance of compression schemes. These factors have directed us to propose a novel compression scheme called locality-aware compression (LoComp), which can be summarized as follows.

1) LoComp provides a data layout aimed at minimally impacting the data locality of applications.
2) In conjunction with the proposed data layout, LoComp effectively manages ECCs and CFs by utilizing a partial write operation. Partial write operations allow access to a byte (or once among bursts) of DRAM data, which is very effective in reducing extra memory access.
3) We also enhanced LoComp for byte (or bit)-level data packing algorithms. Therefore, we propose OBΔI, by extending the Base-delta-immediate (BΔI) compression algorithm [18]. OBΔI shows higher compression ratio for these applications.
4) We further enhanced LoComp by reorganizing CFs through a tradeoff between the compression ratio and DRAM access time. We call this enhanced version LoComp+.
5) To demonstrate the effectiveness of LoComp and LoComp+, we implemented a system level simulation framework by extending GEM5 simulator [19] and DRAMSim2 [20].

The remainder of this paper is organized as follows. First, we review important compression methods and the related ECC schemes in Section II. In Section III, we address the importance of aforementioned issues by showing motivating examples. We discuss LoComp and LoComp+, including their data layout, compression method, and other details in Section IV. Finally, we prove the effectiveness of LoComp through extensive experiments in Section V followed by the conclusion in Section VI.

II. Background and Related Works

In this section, we summarize prior embedded ECC schemes specialized for main memory in four categories. First, we review the concept of embedded ECC schemes and compression in main memory. The next two categories are dedicated to ECC and compression algorithms in Sections II-B and II-C, respectively. Finally, data layout and metadata management issues are addressed as the third category in Section II-D.

A. Compression and Embedded ECC in Main Memory

Fig. 1(a) depicts the flow of a cache line between the LLC and DRAMs. In most systems, the LLC line size is typically 64B. Also, a DRAM array consists of multiple DRAM chips. More specifically, an 8 $\times$8 (8-bit wide) chips DRAM consists of 8 DRAM chips, each of which has 8 data IO pins. In this

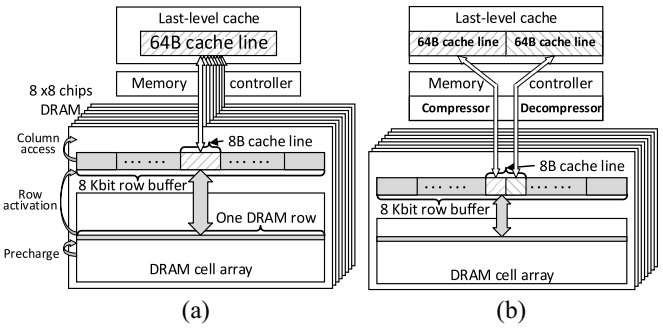


Fig. 1. Data flow and compression of cache lines in DRAM. (a) Data flow of a 64B cache line. (b) Compression example of cache lines in DRAM.

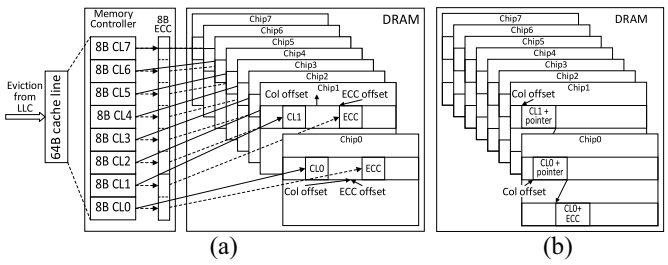


Fig. 2. Data flow and layout for a 64B cache line and ECC in embedded ECC. (a) Deterministic method. (b) Pointer method.

configuration, the 64B of a cache line is formed by 8B from each DRAM. Also, each DRAM chip has an internal buffer called the row buffer [3]. Typically, it is implemented as sense amplifiers and its size is 8 Kbit. When a cache miss occurs in the LLC, a request for the cache line is transmitted to the memory controller through a system bus. And the memory controller conveys DRAM access commands on the DDRx interface. If the requested cache line is in the row buffer of DRAM, called the row buffer hit, the memory controller can access the cache line after column access latency. Otherwise, if the requested cache line is not in the row buffer, called the row buffer miss, the cache line access requires the additional latency of precharge and row activation in a DRAM cell array.

To reduce this latency, hardware compression in the memory controller is a promising approach. The compression is performed in a quantized manner. More specifically, a 64B cache line is compressible if its compression ratio is higher than the threshold. As an example, if both of two consecutive cache lines requested to DRAM are compressible higher than or equal to 50%, two cache lines can be compressed to one cache line and stored to the same row address by one burst write as shown in Fig. 1(b). In addition, if two cache lines have temporal locality in memory read requests, the request for the second cache line causes the row buffer hit. In short, compression changes the layout of cache lines in the row buffer.

In embedded ECCs, the coexistence of cache lines and corresponding ECCs also impacts on the layout of the row buffer. There are two major approaches for the data layout: 1) the deterministic offset approach and 2) the pointer-based approach. Fig. 2(a) shows the deterministic offset approach in case of DRAM write. When a cache line is evicted from the LLC, each 8B (CL0–CL7) of a 64B cache line and the corresponding ECCs are stored to the predetermined offset in a chip. Therefore, the address computation is very simple, but the memory space overhead owing to ECCs cannot be neglected. The memory-efficient pointer-based method shown in Fig. 2(b) uses a pointer to access the leftover directly.

B. Embedded ECC Algorithms

Typical memory systems for server systems adopt single error correction, double error detection (SECDED) ECC algorithms with a separate memory chip to store ECCs. Chipkill-level ECC is another option, especially when data integrity is a critical concern. It basically applies a symbolic correction code [7], [14] and/or a multiple tiers code [7], [8] to handle more error-prone conditions compared to SECDED ECC algorithms. Owing to its nature, chipkill-level ECCs require more redundant areas for ECC parities than SECDED ECC algorithms. Memory systems with chipkill-level ECC algorithms embed ECC parities in the data space for reducing extra ECC parity chips.

On the other hand, area efficiency is a more critical factor in client and/or embedded systems. For this reason, SECDED embedded ECC algorithms are widely adopted, as discussed in [6], [9], [12], and [13]. These are often implemented based on a (72, 64) Hamming code to encode a 64-bit message to a 72-bit codeword with eight redundant bits, where seven bits are used as Hamming parity ($m = 7$) and the remaining one bit is used as even parity.

C. Compression Algorithms for Embedded ECC Schemes

BΔI [18] and frequent pattern compression (FPC) [21] are widely adopted for main memory compression [22]. Compression algorithms based on LZ or Huffman coding have high latency/area overhead but provide high compression ratio. In contrast, BΔI and FPC can be implemented with simple hardware, which takes 1 or 2 cycles for compression or decompression. Therefore, these algorithms are suitable for embedded systems. BΔI exploits the similarity between the words within a cache line. A cache line can be substantially compressed, since BΔI stores a base word and deltas which are the difference between the base word and other words. Recent works have modified BΔI to have more compressible cache lines by using multiple bases [18] or increasing the base size and delta size [14]. FPC also exploits word-level similarity like BΔI. In FPC, the reference pattern is not given by the base word of each line, but the special patterns are predetermined. For identified words to be compressed, it encodes each word as a predetermined prefix and compressed data. FPC has high compression ratio, especially for sign-extended words and words consisting of repetitive bytes. The compression ratios of these algorithms depends on address alignment. More specifically, the similarity between compression chunks is low if the start address of a cache line is misaligned with the start address of the chunk.

Although LoComp is a generalized approach that can be integrated with any main memory compression algorithm, LoComp adopts BΔI and FPC that are well suitable for

embedded systems. Moreover, LoComp provides an extended feature to manage such misalignment to achieve higher compression ratio.

D. Data Layout and Metadata Management for Embedded ECC Schemes

As data layout and metadata management mainly contribute to memory access efficiency, they are often synergistically designed with compression algorithms to achieve globally optimal memory access efficiency. In this section, we first review compression algorithm-independent data layout techniques, and then address data layout techniques customized for specific compression algorithms.

1) Compression Algorithm Independent Data Layout: Data layouts for embedded ECC schemes have been studied to improve specific performance metrics.

Mini-rank [6] focuses on dynamic power consumption, which is one of the critical issues in DIMM-type memory systems. To address this issue, the authors split a memory rank into subranks, which are controlled independently by an additional controller on DIMM. For ECC implementation, they adopted an embedded ECC scheme and data layout with Chinese remainder mapping (CRM) [23] that eliminates the overhead of a division operation.

While Mini-rank focuses on power efficiency, E^3CC [9] concentrates on the bandwidth utilization of DDRx by supporting full-rank architecture. The data layout in the E^3CC architecture is exactly matched to the burst size of the main memory. When a cache line is accessed by the host processor, the corresponding ECC line is simultaneously fetched. Only one out of eight parities in the ECC line is utilized for the cache line, and the remaining seven parities are stored in the ECC cache for future use. E^3CC also tackled the address mapping scheme by employing biased CRM (BCRM), a modified CRM to maintain row locality.

Virtualized ECC [7] proposes two-tier ECC protection for power-efficient chipkill-level ECCs. For this purpose, each tier detects and corrects errors independently. The OS takes charge of address translation for two-tier codes from the physical address of LLC, like virtual address translation. LOT-ECC [8] further generalizes Virtualized ECC by employing multiple levels of localized and tiered error protection to reduce latency. However, it entails larger area overhead than Virtualized ECC.

2) Compression Algorithm Specific Data Layout: The compression-specific data layout requires a more sophisticated design than the compression-independent data layout. In other words, it should consider the location of metadata, which indicates whether the corresponding data is compressed or not.

MemZip [12] tries to reduce the number of write/read (WR/RD) commands by compressing data into subranked memory organization. For instance, if all cache lines are compressed by 50%, the main memory bandwidth utilization can be doubled. It is obvious that MemZip is quite effective in reducing the WR/RD command execution. For this purpose, MemZip organizes the 8 KB row buffer of a memory chip with 112 data cache lines, 14 ECC lines, and a metadata line. Therefore, 16 data cache lines become leftovers because of the ECC lines and metadata lines. The leftovers must be placed in a different row, called an overflow row. There is also extra performance overhead from opening rows multiple times due to the overflow rows.

To avoid the metadata overhead, COP [13] implements the identification of compression with SECDED. Thus, a row buffer of main memory can contain more data cache lines compared to MemZip. However, the main drawback is the undetectable errors of SECDED, with error probability of approximately 7.8×10^{-3}, in the case of $m = 7$ [24]. COP eventually keep aliasing cache lines which caused confusion in compression identification. COP defines a separate address region for leftovers. Each cache line includes a pointer to the leftovers when the cache line is incompressible. In this case, COP requires additional memory access for the leftovers. It also requires the LLC to have a cache tag for pointers to efficiently retrieve or invalidate the pointers. COP may degrade the spatial locality since it places leftovers in a different address region.

FECC [14] place ECCs with compressed data similar to COP, but its target is chipkill-level, which requires larger space for ECCs compared to the aforementioned works. FECC places the protected CF ahead of the corresponding data cache line and encodes CF along with the data for ECCs. With this layout, it is possible to reduce the memory access frequency. The limitation of FECC is the predetermined addressing of leftovers without considering the locality.

III. Motivating Example

In this section, we provide motivating examples about the applicability of compression and the impact of the data layout or data locality in the row buffer of DRAM. Memory compression increases memory capacity without area overhead in modern embedded or mobile systems, which can reduce the storage access that has large latency. For example, in typical SSDs, the huge mapping information of the flash translation layer (FTL) is cached to DRAMs to hide the long latency of NAND flash memories. 4TB SSD [25] adopts 4 GB DRAM for this purpose. However, the increase of DRAM capacity causes the increase of cost as well as power consumption. To mitigate this issue, compression is one of the most appropriate candidates by compressing metadata to be stored to the DRAM cache. On the other hand, compression can become an overhead for many other programs (e.g., multimedia applications), since their data is inherently compressed. In Fig. 3(a), we analyzed the percentage of cache lines with its compression ratio exceeding the threshold of 14.1% for several SPEC2006 benchmarks, bbench for a mobile benchmark, and the FTL benchmark called DFTL [26]. For the compression algorithm, we used BΔI and FPC. When cache line accesses are requested to DRAM, the compressor in the memory controller compresses or decompresses on a cache line basis.

In short, the applicability of a compression scheme is highly dependent on the characteristics of applications, as shown in Fig. 3(a). For instance, libquantum shows a compression ratio less than 15%, while the compression ratios of DFTL and bbench are 89% and 98%, respectively. It is obvious that outstanding compression algorithms will show better

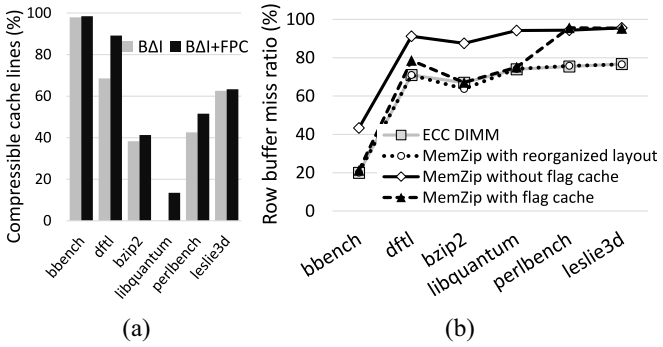


Fig. 3. Compressible cache lines ratio and row buffer miss ratio for some benchmarks. (a) Compressible cache lines ratio. (b) Row buffer miss ratio.

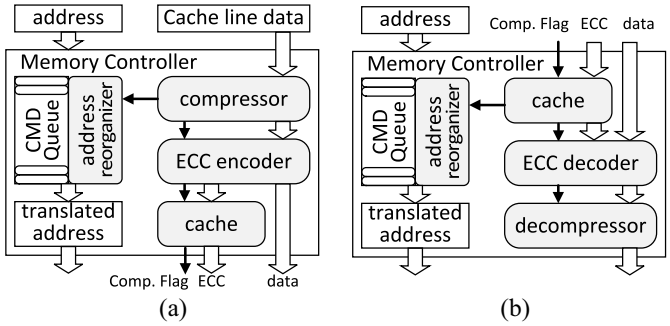


Fig. 4. Memory access flow with the controller of LoComp. (a) Write flow. (b) Read flow.

performance, since they generate less leftovers. Many previous works have already discussed the importance of compression engines [22].

Next, we analyzed the impact of the data layout in embedded ECCs from a spatial locality perspective. Typically, the total size of the row buffer of DRAM chips is 8 KB, which corresponds to 128 cache lines. If the application has a strong spatial locality, 128 cache lines in a row buffer can be fetched with a single open row operation. However, spatial locality is damaged by the metadata in embedded ECCs because some cache lines have to be moved to overflow row(s), which increases row activation and the row buffer miss ratio. In Fig. 3(b), two solid lines (square for ECC DIMM and rhombus for MemZip, a deterministic offset approach) clearly show the impaired spatial locality due to the embedded ECC. The miss rate indicated by the solid line with square markers is optimal, since the inherent spatial locality of each application is not disturbed by the metadata, such as ECCs in ECC DIMM. Reduced spatial locality in embedded ECC is inevitable due to the leftovers which do not satisfy the given compression threshold. The row buffer miss ratio in MemZip decreases by an average 18% compared to the optimal ratio. A similar trend can be observed with the pointer-based method, even though the row buffer miss ratio is lower compared to the deterministic offset method.

If we reorganized the layout to place the data cache line and corresponding metadata in the same row, the row miss ratio drops to a level similar to the optimal (difference under 3%), as shown by the dotted line with circle markers in Fig. 3(b). Finally, this example shows the importance of managing metadata such as CFs. The dashed line with triangle markers shows the impact of the flag cache to exploit the temporal locality of CFs. For applications such as bbench, bzip2, and libquantum, the row buffer miss ratio of MemZip is comparable to that of ECC DIMM by compensating the impaired spatial locality with the temporal locality. But it also shows that the flag cache is effective only for applications that show strong temporal behavior.

To summarize, data layout is a crucial design parameter of embedded ECC systems for the spatial locality of DRAM row buffer as well as compression ratio. Temporal locality exploitation using flag cache can further enhance the performance of the system, especially when the applications show highly temporal behavior. The proposed LoComp synergistically integrates these three design parameters and its details will be discussed in Section IV.

IV. LoComp Design

LoComp is a generalized approach that can be integrated with any memory controller. In Fig. 4, we illustrate the read and write flows of a memory controller equipped with LoComp, respectively. The write flow of LoComp is shown in Fig. 4(a). Once a host processor issues a write request, the memory controller holds the request in a queue. While the request is in the queue, LoComp compresses a cache line to make space for ECC, similar to COP. In the compressor, LoComp enhances the BΔI algorithm to make more compressible blocks for cache lines containing the packed data structures or random variables. At the same time, LoComp performs address translation to identify the location of the data to be placed. LoComp provides the data layout with minimal impact on locality and fits a compressed cache line, ECC, and CF into a same row buffer of DRAM. In contrast, existing approaches such as COP and MemZip break the row locality and increase DRAM access time. After these two operations, the ECC encoder generates ECCs and both CFs and ECCs are cached in LoComp. In contrast, only one of them is cached in existing approaches. In the final step, all these data are written sequentially to the corresponding memory location. At this time, LoComp exploits the partial write of a single byte unit to reduce read–modify–write operations for 1-byte CFs through controlled chip select (CS) signals, while the granularity of data masking is 8 bytes in conventional DRAMs. Note that LoComp is implemented in a pipelined fashion. Therefore, its latency penalty is marginal when it handles burst requests. The read behavior, which is the reverse of write flow, is also shown in Fig. 4(b).

A. ECC Algorithm

LoComp does not employ the chipkill ECC algorithm but the SECDED algorithm for area efficiency in embedded systems. Table I shows the failure in time (FIT) comparison between (72, 64) SECDED for LoComp and (136, 128) In-memory SECDED for commercial LPDDR4 [27]. The simplified error model assumes randomness of bit flips, and the probability of a single-bit error in one hour is referred from [28]. (72, 64) SECDED reduces FIT by 56% compared

TABLE I
ERROR PROBABILITY ACCORDING TO ECC ALGORITHMS

Error probability of single-bit in a hour [31]	Failures per 1 billion hours (FIT) in 1GB DRAM				
	without ECC	(72, 64) SECDED	(136, 128) SECDED	(532, 512) BCH	(542, 512) BCH
MIN : 2.5e-11	1.93E+08	2.14E-01	3.85E-01	6.54E-09	2.33E-17
MAX : 7.0E-11	4.52E+08	1.68E+00	3.02E+00	1.44E-07	1.43E-15

to (136, 128) SECDED of LPDDR4 that is popularly used in embedded systems for low power. Since (72, 64) SECDED of LoComp shows lower FIT than the commercial in-memory SECDED of LPDDR4, it ensures that DRAM is sufficiently reliable for embedded systems.

Although the raw FIT that corrupts adjacent column cells is 1.4×10^{-4} times lower than that of a single-bit error [8], it is possible to extend LoComp to support ECC algorithms for multibit error correction due to its flexibility. For instance, LoComp can provide (532, 512) or (542, 512) BCH codes for two-bit or three-bit error correction, respectively, in a 64B cache line. Their FIT is lower than (72, 64) SECDED as shown in Table I. However, BCH requires long decoding latency and hardware overhead to solve an error location polynomial. LoComp can also provide an interleaving technique [29], [30] for burst error correction up to 8-bit per a 64B cache line because interleaving is an orthogonal technique to LoComp.

B. Compression Algorithm

The BΔI compression algorithm is designed for word-level aligned data representation. Fig. 5 shows the typical BΔI format, where the base is the first 4B (or 8B) data chunk of each cache line. Every 4B (or 8B) chunk succeeding the base can be compressed as 1B Δ by computing the difference between the base and each chunk. The basic assumption in this algorithm is that the dataset is aligned in word level, which is quite reasonable for modern computer architectures. However, the programming model in many embedded systems allows misaligned data representation by packing data for memory efficiency [31]. For instance, structures and/or structure arrays are good packing candidates. The top example in Fig. 6(a) shows that the start address of the Structure array is misaligned with the compression chunk due to the 1B packed variable. This misalignment diminishes the similarity between a base word and delta words in BΔI. To resolve this problem, OBΔI of LoComp considers an offset field prior to the base field, as shown in Fig. 6(b). To adjust the misalignment, the size of offset field varies from 1B to 3B.

Another effective feature of OBΔI is data masking for highly incompressible data. This feature is effective even for word-level aligned data representation. OBΔI also considers the case where a structure is aligned but has a random variable, as the second example depicted in Fig. 6(a). The Structure array will have low similarity with this variable owing to its randomness. A simple example of this case is the array for time logging, where the LSB of logged time is random. On the other hand, the similarity of the MSB excluding the LSB is high. Eventually, more structures turn into compressible data if the LSB is masked. For byte-level masking, OBΔI provides the second format shown in Fig. 6(b). Though a masked byte

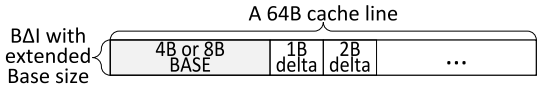


Fig. 5. Typical BΔI formats.

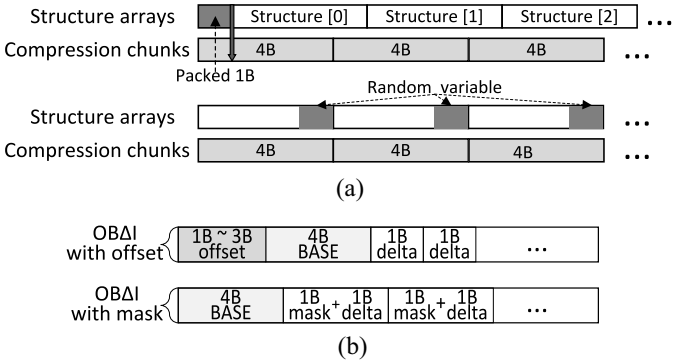


Fig. 6. OBΔI formats and compression examples. (a) Examples for the packed structure, and the structure including a incompressible random variable. (b) Optimized BΔI formats.

remains in a delta field, its compression efficiency improves when the MSBs of the 4B chunks in a cache line show similarity. The performance improvement of OBΔI will be discussed in detail in Section V-B. Although OBΔI enhances compression ratio, OBΔI increases comparators in the compressor by four times compared to single-base BΔI, but has the same hardware overhead as multiple-base BΔI [18]. In addition, this hardware overhead of the compressor can be reduced by 75% by pipelined implementation for a 64-bit system bus. The decompressor of OBΔI requires marginal hardware overhead, since each algorithm selectively operates with a common decompressor.

LoComp provides another compression algorithm, FPC for achieving higher compression ratio. LoComp uses a 3-bit prefix to indicate the compressed formula of every word, which is similar to the original FPC introduced in [21].

To distinguish the compression formats, LoComp requires slightly higher compression ratio compared to conventional BΔI and FPC. While a 64B cache line must be compressed to less than or equal to 56B to have the space for 8B ECC in conventional BΔI or FPC, LoComp requires a cache line to be compressed to less than or equal to 55B and utilizes 1B to store compression information. More specifically, the 1B compression information consists of the MSB indicating the compression algorithm and the remaining bits indicating the submechanism of each compression algorithm. LoComp executes both compression algorithms in parallel and takes the higher compression result, like other compression schemes.

C. Data Layout in DRAM Row Buffer

LoComp provides the deterministic layout for compressible cache lines regardless of the compression ratio for simple address calculation. Therefore, it requires a total redundancy of 14.3%, which includes 12.5% and 1.8% of ECCs and CFs, respectively. Moreover, LoComp provides the specialized data

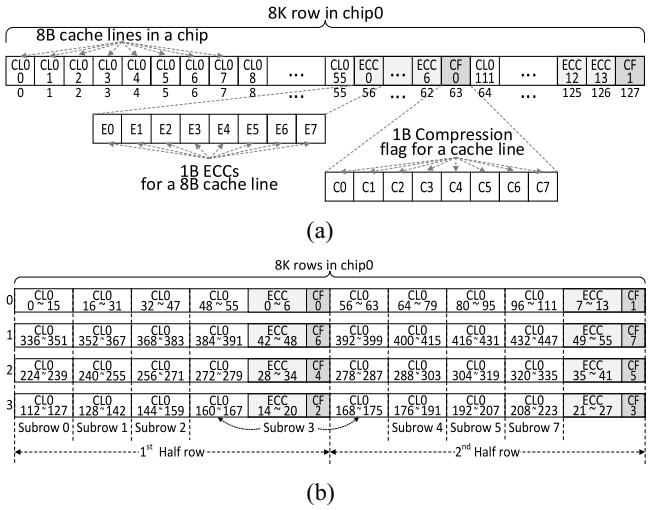


Fig. 7. LoComp physical layout for ECC and CF. (a) LoComp physical layout in a row. (b) Sequence of cache lines in a chip0.

layout to improve performance by considering the following two localities.

First, the data layout should not break the physical address sequence of the cache line to prevent redundant row activations. However, the address translation of embedded ECC schemes often breaks this sequence. Typically, this address translation raises the hardware overhead, such as a divider. If the number of cache line in a row is 112, it is not a power of two. To eliminate the division overhead, Mini-rank uses CRM [23], which requires only two modular calculations instead of division. However, it impairs row locality since the cache lines of contiguous physical addresses exist in separate rows. To resolve this issue, BCRM [9] adds a bias factor to CRM. LoComp follows BCRM to calculate primitive addresses of the cache line.

Second, to avoid impairing row locality for all memory accesses, including ECCs or CFs, LoComp places a data cache line with its corresponding ECC and CF in the same row. With this layout, it is possible to fetch each data cache line with a single open row, even in the worst case where these three types of access occur consecutively. Fig. 7(a) depicts a row of chip0 in an 8 × 8 chips architecture, shown in Fig. 2. A row consists of 112 cache lines, 14 ECC lines, and two CF lines, as shown in Fig. 7(a). CL0 represents the first 8B data among a 64B cache line in a row of chip0, while ECC represents the 8B ECC which are eight parities corresponding to eight CL0. CF0 and CF1 correspond to the CFs—each for 56 sequential cache lines, respectively.

CRM and BCRM map a given address x into a pair of integers (u, v) with $0 \le x \le p{\cdot}m-1$, only if p and m are coprime. Assuming that the address scheme is row:bank:column and a full address is d, the generalized formulas of BCRM are as follows:

$$x = d \gg \mathrm{S}, \quad o = d \bmod 2^S \tag{1}$$

$$\begin{aligned} b(x) &= (x - (x \bmod m)) \bmod 2^B \\ r(x) &= ((x - (x \bmod m)) \gg B) \bmod 2^R \\ c(x, o) &= ((x \bmod m) \ll S) + o \end{aligned} \tag{2}$$

where 2^S is the cache line number in subrow, 2^B is the number of bank, and 2^R is row number in a bank. The subrow address x, and the offset address o in a subrow is defined as (1). In (2), $b(x)$ is the bank address and $r(x)$ is the row address, where m is the coprime number with $2^{(B+R)}$ and $c(x, o)$ is the primitive column address. For example, assuming that the memory consists of one bank with four rows as shown in Fig. 7(b), a row consists of seven subrows and each subrow has 16 cache lines. In other words, 2^R is 4 and m is 7 in (1). Although the sequence of rows is changed $(0 \to 3 \to 2 \to 1)$, the sequence has no impact on row locality and the sequentiality of cache lines within a row lasts.

In addition to maintaining the locality of cache line with BCRM, LoComp extends BCRM formulas as defined in (3) to prevent damage to the row locality of memory access, including ECCs or CFs. Each column address for data cache line ECC and CF is $dc(x, o)$, $ec(x, o)$, and $fc(x, o)$, where C, E, F, and T are the number of data cache lines, ECC lines, CF lines, and total lines in a row, respectively. C^H is the number of data cache line in a half row and 2^P is the number of ECCs in an ECC line

$$\begin{aligned} dc(x, o) &= \begin{cases} c(x, o) & \text{if } c(x, o) < C^H \\ c(x, o) + E^H + F^H & \text{otherwise} \end{cases} \\ ec(x, o) &= \begin{cases} (c(x, o) \gg P) + C^H & \text{if } c(x, o) < C^H \\ (c(x, o) \gg P) + C + F^H & \text{otherwise} \end{cases} \\ fc(x, o) &= \begin{cases} T^H - 1 & \text{if } c(x, o) < C^H \\ T - 1 & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Although the row size of commercial DRAMs is 8K, LoComp aligns data, ECC, and CF lines with 4K in 8K rows to support fine-grained row architecture, as shown in Fig. 7(b). Recent works have proposed different implementations of fine-grained DRAM architecture to reduce latency or energy [32], [33]. Basically, these prior works reduced the row size to 4K. In embedded or mobile systems that are targets of LoComp, latency and energy are important metrics. Therefore, LoComp aligns the data layout with 4K to make it suitable for fine-grained DRAM architecture.

The modular calculation of 2^n involves the selection of LSB bits in a binary word of length n. The simple implementation of this is bit masking. The modular calculation of 7 is simplified by adders and some registers [34]. Moreover, the shift operation is implemented by a bitwise connection and the add operation is a bitwise concatenation of the modular result and o as the bottom equation of (3). Since other add operations are arithmetic calculations with constants, we can implement address mapping logic with small HW overhead.

D. Request Management

Each cache line requires a 1-bit CF which indicates whether a cache line is compressible or not. LoComp protects CF with (8,4) SECDED for reliability. For this purpose, LoComp utilizes an 8-bit codeword followed by three reserved bits which are padded by zeros. Also, m is chosen as 3, since the (8, 4) Hamming code provides lower undetected error probability than when m is 2 [24]. Fig. 8(a) shows that a 64B CF line includes flag bits for 56 cache lines. LoComp supports both four chips with narrow bandwidth organization, as well as two

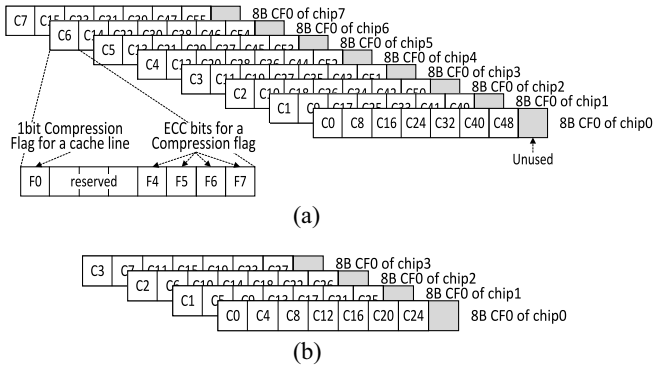


Fig. 8. CF layout. (a) For 8 ×8 chips. (b) For 4 ×8 chips.

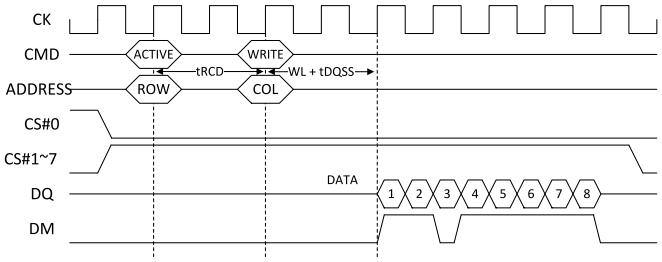


Fig. 9. Write timing for partial write of CF.

chips with wide bandwidth organization to tradeoff power efficiency and performance. Fig. 8(b) shows the layout of CF for four-chip organization. If a cache line is compressible, a 64B cache line is split into two 32B burst accesses with a 32-bit channel width and requires one CF line access.

Even if a cache line is incompressible, LoComp provides the access efficiency of metadata using two request management methods. First, LoComp manages the metadata with two separate small caches. If a read request is issued for incompressible data, LoComp accesses three columns to read a CF line, an ECC line, and a data cache line in the worst case. When an incompressible read request enters the controller in 8 × 8 chips organization with a 64-bit data bus, LoComp has to consume the unnecessary bandwidth (56B ECC and 63B CFs) for other cache lines. This is because the default burst length of commercial DDR3 is eight. To avoid this inefficiency, LoComp includes small ECC and flag caches in the memory controller, as shown in Fig. 4. The size of the ECC cache is 256B and consists of four blocks, each with the size 64B. The size of the flag cache is 252B and it has 36 blocks of 56-bit size. The size of a block in the flag cache is exactly matched to the number of decoded flag bits in a 64B CF line. To prevent performance degradation, the flag cache has more blocks than the ECC cache because LoComp always accesses the CF for every data request, unlike ECCs.

However, when a write request is incompressible and ECC or flag cache miss occurs, LoComp should perform read–modify for the fetched ECC or CF line and write each evicted line to the main memory. To enhance performance by reducing unnecessary access, LoComp first uses the data mask (DM) signal of a DDRx specification. Each DM bit can mask data bytes (DQ) for any cycle of the burst, but DM can only be used for the ECC partial write and not for CF because the minimum

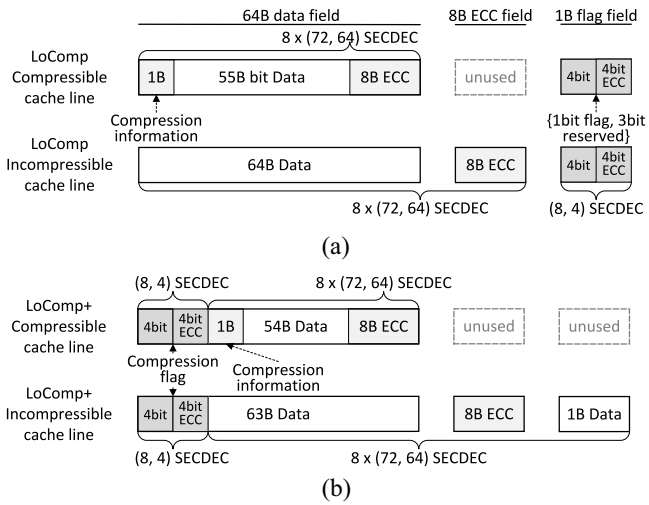


Fig. 10. Logical layout for a cache line. (a) For LoComp. (b) For LoComp+.

masking granularity is 8 bytes in the 8 × 8 chips. To reduce the granularity of the partial write without additional DM pins, LoComp exploits the CS signal. In conventional DRAMs, the CS signal is active-low and always in a low state. In contrast, LoComp controls the CS and DM signal for the 1B partial write. Fig. 9 illustrates the timing of a single request to write 1B CF of chip0 partially with CS and DM. First, LoComp disables the CS signal of chips except chip0 to mask chip1 to chip7 during Active and WR/RD commands. LoComp enables DM of chip0 to mask all data except the third data during the burst transfer. The implementation overhead for controlling the CS signal is small because CS does not require high-speed IO pads, such as DQ or DQS pads.

E. LoComp+: Enhanced LoComp for CF Management

When a cache line is compressible and a flag cache miss occurs, LoComp needs to access DRAM twice—one for a CF line and one for a data line including ECC. To eliminate the access for CF, we propose a modified LoComp data layout called LoComp+. A few previous works have attempted to reduce the overhead of CF access. For instance, COP relies on the probability of the Hamming code for this purpose. However, it cannot avoid the inherent overhead for managing the aliasing cache lines in LLC. FECC avoids overhead by placing the ECC flag in the data field. Although this feature enables FECC to access the memory only once for compressible cache lines, FECC has two critical limitations. One is the predetermined addressing of leftovers without considering locality. The other is the need to fetch leftovers from the memory controller to the LLC, which increases the system bus traffic and does not support reliable swap operation directly with I/O devices.

We compare the logical data layouts of LoComp and LoComp+ in Fig. 10. The major change in LoComp+ compared with LoComp is the location of the 1B flag field. More specifically, the CF is located at the first LSB in LoComp+. This relocation of CF requires a higher compression ratio for the compressible cache line, since 1B in the data field has to be used for CF. However, the increase in compression ratio

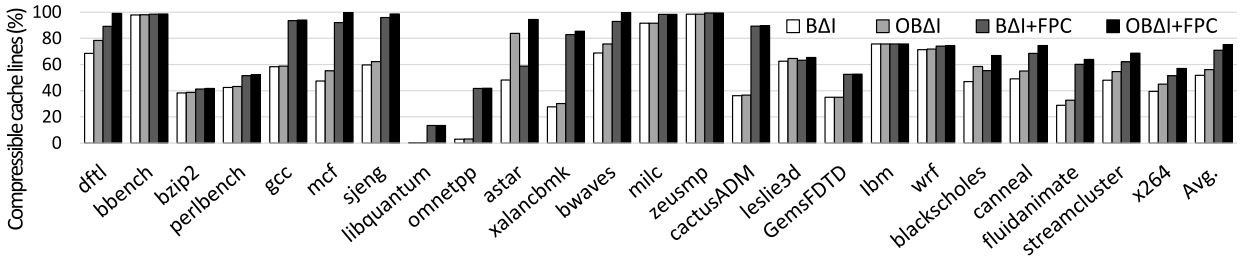


Fig. 11. Cumulative compressible ratio with compression schemes.

TABLE II
SIMULATION PLATFORM CONFIGURATION

Processor	ARM 4-Core 2 GHz
L1 cache	Icache : 32 KB/4-way, Dcache : 32 KB/4-way 64B cache lines, 2-cycle hit latency
L2 cache	512 KB/8-way 64B cache lines, 20-cycle hit latency
DRAM controller	Queue Depth : 32, Queuing per rank open page policy, 64-bit data bus
DRAM	DDR3-1600, 2 Gb per chip CL-tRCD-tRP : 11-11-11, BL : 8 8 banks per chip 32768 rows and 1024 columns per row

TABLE III
WORKLOADS

Suits	Workloads
SPECint 2006	bzip2, perbench, gcc, mcf, sjeng libquantum, omnetpp, astar, xalancbmk
SPECfp 2006	bwaves, milc, zeusmp, cactusADM leslie3d, GemsFDTD, lbm, wrf
PARSEC 3.0	blackscholes, canneal, fluidanimate streamcluster, x264
FTL application	dftl
Mobile application	bbench

for the compressible line is marginal, meaning that the overall ratio of compressible ratios is rarely affected. With this layout, it is possible to fetch the compressible cache line with a single memory access. In the case of an incompressible cache line, LoComp+ places the 1B leftover of 64B data in the flag field, while LoComp places the flag and its ECC in the flag field. Therefore, there are four blocks of 64B in the flag cache of LoComp+, which is different from the number and size of blocks in LoComp. Although LoComp+ requires higher compression ratio and has fewer blocks for the same cache size, performance improvement through metadata relocation is significant. The detailed simulation results will be presented in Section V-C.

V. EXPERIMENTS

A. Experimental Setup

We evaluated LoComp in the GEM5 simulation environment [19]. We used the ARM core configuration of GEM5 for mimicking embedded or mobile systems. We have integrated DRAMSim2 [20] into GEM5 to accurately evaluate the impact of LoComp on DRAMs. The simulator configuration used in our experiments is summarized in Table II. We implemented the proposed compression algorithms in the DRAMSim2 wrapper of GEM5 and modified the address transaction of DRAMSim2 to verify the improved performance of LoComp. We used the DRAM timing parameters of Micron DDR3 datasheet [35].

For our simulations, we employed various workloads which are memory-intensive applications. As shown in Table III, there are 23 workloads from SPEC2006 benchmarks [36] with reference inputs, as well as the PARSEC benchmark [37] with larger inputs than a medium-sized input. We also used the DFTL workload to evaluate LoComp with SSD applications. DFTL is one of the FTLs using page-level address mapping. We also used bbench [38] to appreciate the performance of LoComp in an Android environment. The bbench is a benchmark that tests a browser's page rendering performance. For each workload, we simulated over 5 million memory requests for each benchmark from the boot time of the Ubuntu Natty or Android ICS OS. To avoid the initialization effect owing to the compressible all zero data or the boot process, the performance statistics are calculated for the last 4 million requests. Although all zero data with high compression ratio is favorable to LoComp, we discarded first 1 million or more requests to obtain practical compression results. The 4 million requests were taken from 0.4 billion to 11 billion CPU cycles, depending on the benchmarks.

B. Evaluation of Compression Algorithms

Whenever DRAM requests enter the DRAMSim2 wrapper, the simulator determines whether the request is compressible or not according to compression algorithms. Fig. 11 shows the cumulative compressible ratio using each compression algorithm. Compressibility improved by an average 5% for the whole benchmarks by adjusting misaligned compression targets. In particular, there is improvement of up to 10% in the DFTL workload because it has lots of packed data structures and mapping tables. Some benchmarks showed poor compressibility during simulations. In bzip2, compressibility was less than 50% because bzip2 was the compression application and half of the data used in memory was already compressed by this application. Libquantum and omnetpp showed less compressibility compared to other benchmarks because the incompressible data for the quantum computing algorithm and Ethernet network events were accessed, respectively, by the CPU during the evaluation time. Although high compressibility is important for LoComp to improve performance, another purpose of LoComp is to further improve the performance of benchmarks with poor compressibility by reducing memory access. Fig. 12 shows the stacked ratio of compressible cache

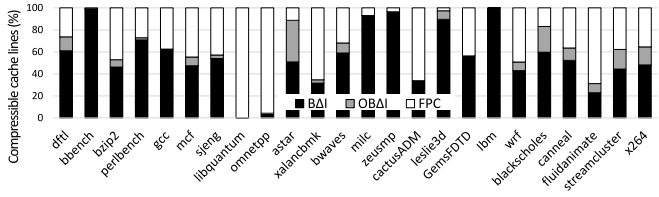


Fig. 12. Breakdown of compression scheme for different benchmarks.

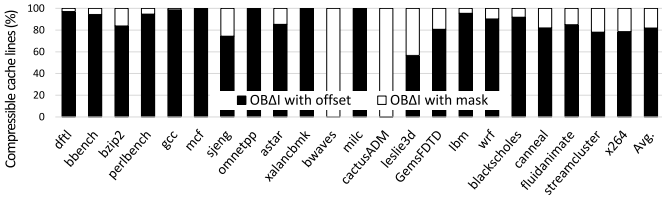


Fig. 13. Breakdown of OBΔI for different benchmarks.

lines for each compression algorithm except all zero patterns. Since all compression algorithms can compress the cache line of all zero patterns, it is meaningful to analyze compression algorithms by excluding all zero patterns. The proportion of cache lines compressed by BΔI, OBΔI, and FPC are 54%, 8%, and 38%, respectively. Fig. 13 shows the stacked ratio of compressible cache lines for the two optimizations of OBΔI, except for the benchmarks that have no improvement with OBΔI. Among the increased compressible cache lines by OBDI, the improvement due to optimization with the offset averaged 82% and the improvement using the mask was 18%.

Since LoComp+ requires 1B extra free space than LoComp to place the encoded CF within data fields, it increases the minimum threshold of the compression ratio from 14.1% to 15.6%. Fig. 14 shows that LoComp+ has 0.8% fewer compressible cache lines than LoComp. The slight difference of 0.8% did not affect performance. In short, LoComp+ can enhance performance by eliminating CF access in the case of compressible cache lines. As the exceptional application, the compressible ratio decreased by 7.5% in the omnetpp benchmark. For this benchmark, LoComp+ decreased performance by 10% compared with LoComp.

C. Performance Results

For the performance evaluation, we compared six SECDED ECC schemes. The two ECC schemes are ECC DIMM and E^{3}CC. ECC DIMM showed the best performance since the inherent locality of each cache line was not disturbed by the metadata. Therefore, the DRAM access time of ECC schemes was normalized to ECC DIMM. E^{3}CC is an embedded ECC scheme that considers locality. Other two schemes were COP and MemZip, which exploited compression. We included these schemes in the experimental group to evaluate the differences in performance according to data layout and request management. The remaining methods are our proposed LoComp and LoComp+. The ECC schemes except ECC DIMM adopted the cache for ECCs or CFs. E^{3}CC includes the 512B ECC cache, which is the smallest cache size among previous studies. To evaluate the schemes under the same conditions, we set the

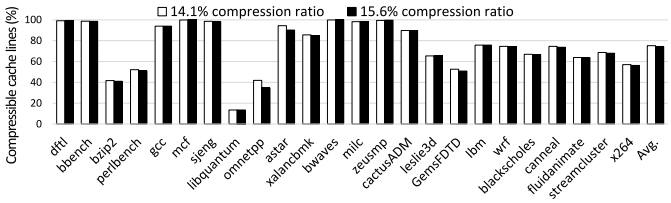


Fig. 14. Proportion of compressible cache lines for 14.1% and 15.6% minimum compression ratio.

maximum cache size for each scheme to 512B, and the same compression algorithms were applied to these ECC schemes. COP should keep L1, L2, and L3 valid bit blocks to find an empty ECC entry. Therefore, we assigned part of the cache for valid bit blocks and the rest excluding valid bit blocks in the 512B cache is used for ECC entries. We evaluated MemZip with the 512B flag cache because MemZip utilizes only one metadata cache. LoComp and LoComp+ include the 256B ECC cache and 252B flag cache. We decided on the cache size in a heuristic way to obtain the best performance. In addition, we evaluated the ECC schemes under the environment where ECC schemes cannot utilize the LLC. We already mentioned why the request manager or the compression engine for ECCs and CFs cannot be located with the LLC in Section I.

1) DRAM Access Time: Fig. 15 shows the measured DRAM access time of the selected benchmarks. Compared to ECC DIMM, E^{3}CC increased DRAM access time by 68% for the whole benchmarks, while COP and MemZip increased it by an average of 81% and 110%, respectively. Even though COP and MemZip utilized compression algorithms, these schemes had greater overhead than the pure embedded ECC, E^{3}CC, because some applications have compressible cache lines less than 50%, as well as break the row locality. In contrast, LoComp and LoComp+ showed the least performance degradation, at 48% and 33%, respectively. More specifically, LoComp+ showed DRAM access time similar to ECC DIMM in dftl, bbench, mcf, sjeng, bwaves, and milc benchmarks. This indicates that LoComp+ can maintain both data reliability and execution time similar to ECC DIMM without additional ECC chips. Interestingly, LoComp and LoComp+ had lower access time compared to ECC DIMM, at 22% and 26%, respectively, in gcc benchmarks, since their reorganized data layout accelerated the bank interleaving while maintaining row locality. In contrast, COP increased DRAM access time by over three times compared to ECC DIMM because it significantly impaired the locality of a data sequence to access ECC entries.

2) Performance Sensitivity to Compression Ratio: Fig. 16(a) shows the proportion of compressible cache lines and DRAM access time normalized to ECC DIMM. The line graphs of COP and MemZip indicate no correlation between the portion of compressible cache lines and DRAM access time because these schemes have long DRAM access time for several applications. This long access time is attributed to their breaking of the row locality while accessing leftovers and/or metadata. MemZip showed exceptional performance degradation of up to three times to access the extra row,

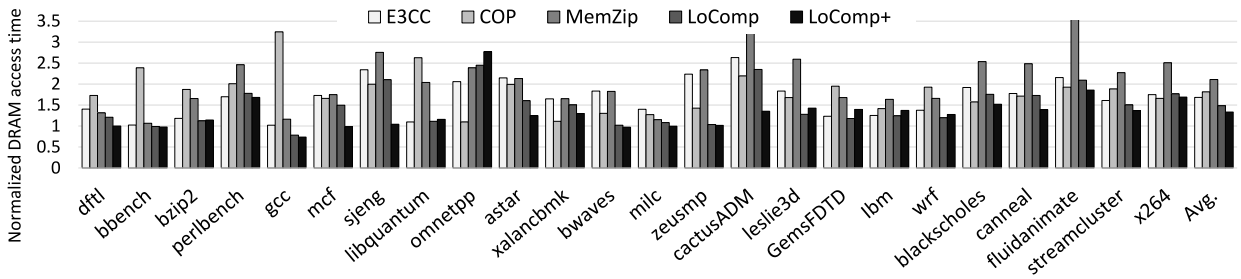


Fig. 15. Comparison of DRAM access time normalized to ECC DIMM between LoComp and prior ECC schemes.

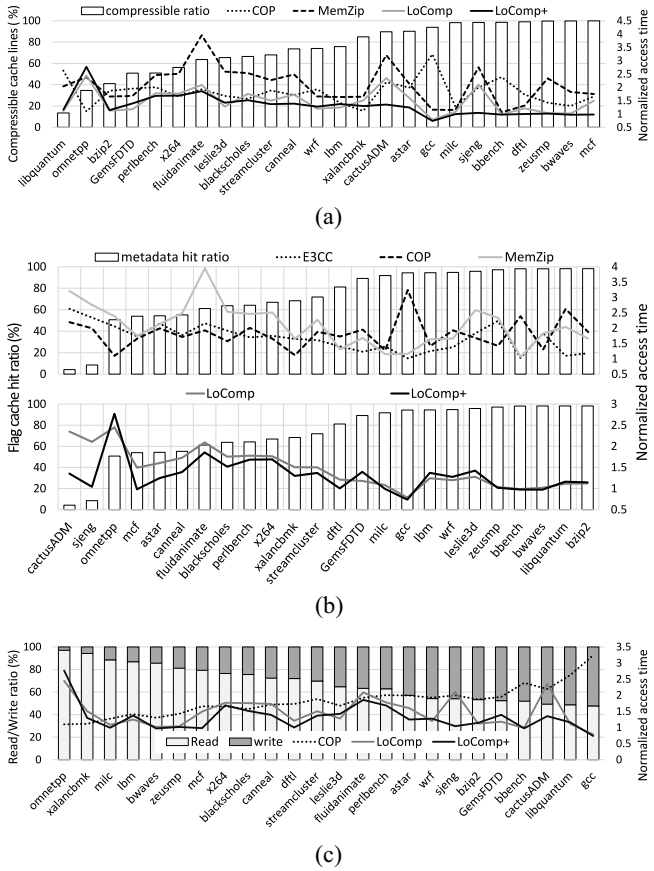


Fig. 16. Effect of compressibility, flag cache, and request pattern on performance. (a) Effect of the portion of compressible cache lines. (b) Effect of the hit ratio of flag cache. (c) Effect of the request pattern.

breaking locality even when benchmarks have high compressibility, such as cactusADM and sjeng. As a noticeable case, the fluidanimate benchmark has low compressibility of less than 65% and MemZip often accessed the data row and separated the extra row alternately. Therefore, MemZip has 3.5 times longer access time. On the other hand, the accessed data were within the same row under the data layout of LoComp. The line graphs of LoComp and LoComp+ gradually decreased as compressible cache lines increased, except for libquantum and bzip2 benchmarks. As expected, these benchmarks had fewer compressible cache lines, but LoComp and LoComp+ showed access times similar to benchmark results that were compressible by at least 80%. This is because these benchmarks have high locality.

3) Performance Variation With Cache and Request Pattern: Fig. 16(b) plots the hit ratios of a flag cache and DRAM access times of each ECC scheme. The hit ratio is one of the temporal locality indices. Although COP has an ECC cache and MemZip has a metadata cache, these schemes break the spatial locality when they access the physical address of data, ECC and CF sequentially. This is why the line graphs of COP and MemZip were not correlated with the cache hit ratio. E^3CC has moderate correlation with the hit ratio of the flag cache because it has only an ECC cache and the number of blocks in the ECC cache is less than the number of flag caches. LoComp and LoComp+ were significantly correlated with the hit ratio of the flag cache, except for the benchmarks with high compressibility, such as mcf, astar, and dftl. LoComp+ showed enhanced performance compared to LoComp for benchmarks with flag cache hit ratio less than approximately 50%. If the hit ratio increased, LoComp had slightly less DRAM access time compared to LoComp+ because of the high hit ratio within the flag cache. On the other hand, in cactusADM and sjeng, LoComp+ significantly reduced access time compared to LoComp because these benchmarks have high compressible ratio, and LoComp+ does not have to access CF lines for the compressible cache line. As shown in these applications, even when the application has low temporal locality, LoComp+ can overcome this drawback with high compression ratio and compression-aware data layout in order to improve performance.

LoComp and LoComp+ were not affected by a request pattern as shown in Fig. 16(c). They increased access time exceptionally in the omnetpp benchmark. It is not that the proposed schemes were affected by a request pattern but that both the compressible ratio and hit ratio of the flag cache were less than 50% in these benchmarks. However, COP decreased performance as the ratio of write increased. This is because COP needed to retrieve the existing pointer for every write request to determine the address of an ECC entry for an incompressible cache line, or to invalidate the existing pointer for a compressible cache line.

D. HW Overhead and Power Results

We synthesized the logic of compression and address translation using Synopsys design/power compiler in 32-nm technology. The area of the compressor/decompressor for OBI and FPC was 31944 μm^2, and their power consumption was 20.01 mW at 1 GHz. The area and power consumption for address translation of each scheme are less than 2%

TABLE IV
AREA AND POWER OVERHEAD NORMALIZED TO ECC DIMM

	DRAM controller		DRAM		Access
	Area	Power	Area	Power	time
E^3CC	↑0.1%	↑0.6%	↓11.1%	↓7.6%	↑68%
COP	↑0.6%	↑2.6%	↓11.1%	↓6.5%	↑81%
MEMZIP	↑0.6%	↑2.6%	↓11.1%	↓5.6%	↑110%
LoComp	↑0.6%	↑2.6%	↓11.1%	↓8.0%	↑48%
LoComp+	↑0.6%	↑2.6%	↓11.1%	↓8.8%	↑33%

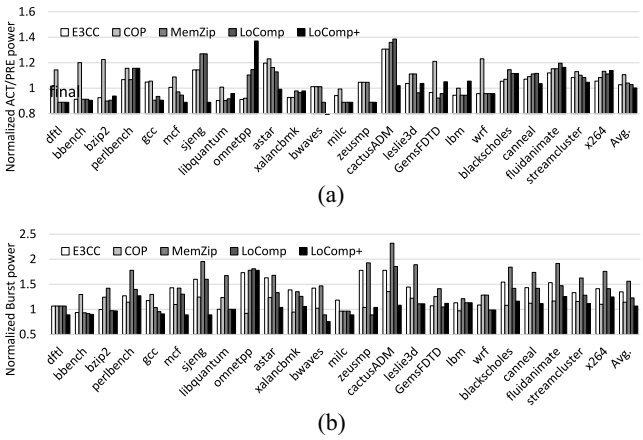


Fig. 17. Normalized power consumption in detail. (a) Normalized ACT/PRE power consumption. (b) Normalized Burst power consumption.

of the compressor/decompressor. We also estimated the area and power consumption of the cache using CACTI 6.5 [39]. The area of the 512B cache was 6335 μm^2, and its power consumption was 5.55 mW in 32-nm technology. Table IV summarizes the hardware overhead and performance improvement of LoComp. Each value is normalized to ECC DIMM. The total area increased for DRAM controller of LoComp was 38666 μm^2, which was less than 1% compared to the 6 mm^2 of a commercial 1-channel DRAM controller including PHY. The area of a commercial DRAM controller was derived with 32-nm processor [40]. And the total power overhead was 26.05 mW, which was about 2.6% compared to 1000 mW of ARM cores in the DRAM access behavior model [41]. For fair comparison with the prior works, we equally set the cache size and applied the same compressor for each scheme, while E^3CC had only the cache overhead. In off-chip DRAM side, since ECC DIMM only required an additional ECC chip, other schemes reduced the DRAM chip array by 11.1% and eliminated the refresh power for the ECC chip. DRAM power indicates the sum of the burst, bank activation (ACT)/precharge (PRE), and refresh power for each scheme.

LoComp and LoComp+ had less performance degradation as well as reduced DRAM power as shown in Table IV because these ECC schemes reduced redundant access for leftovers and/or metadata. Most especially, power efficiency is in ACT and burst operations with the power breakdown. Fig. 17(a) plots the ACT/PRE power consumption of each ECC scheme normalized to baseline, which is the power of ECC DIMM. E^3CC, COP, and MemZip have average power consumption of 3%, 11%, and 4%, respectively, over the baseline. In contrast, LoComp shows the same ACT/PRE power consumption as E^3CC. As expected, E^3CC and LoComp consumed less power to activate the row of DRAM compared to COP and MemZip because E^3CC and LoComp did not impair the original locality of a data sequence. LoComp+ further reduced ACT/PRE power consumption to baseline. Fig. 17(b) shows the burst power consumption of each ECC scheme normalized to the baseline. The average power consumption of E^3CC, COP, and MemZip for the whole benchmarks was 35%, 14%, and 56% greater, respectively, than the baseline. In contrast, LoComp and LoComp+ consumed relatively less burst power, at 23% and 7%, respectively, over the baseline. This is because the cache utilization of LoComp and LoComp+ was optimized for both leftovers and metadata. Moreover, LoComp and LoComp+ decreased the write granularity to eliminate the read–modify–write operations, as described in Section IV-D.

VI. CONCLUSION

DRAM reliability is required not only in server systems or data centers, but also in embedded systems, such as an enterprise SSD. In this paper, we proposed a novel ECC scheme using compression methods, which overcomes drawbacks such as damage of locality and costly read–modify–write operations for redundant access. The proposed LoComp leveraged the benefits of compression, data layout for locality, and metadata management to improve performance with the embedded ECC scheme. These three orthogonal factors make LoComp an effective solution for DRAM reliability and performance issues in embedded systems. In addition, LoComp allows 10% more compression cache lines by optimizing the typical compression engine in dftl benchmarks. The performance evaluation with contemporary ECC schemes shows how the compressible ratio, cache hit ratio, and request pattern affect DRAM access time. In the experiment, LoComp increased DRAM access time by 48% normalized to the baseline. However, previous studies have shown increased overhead, relatively from 68% to twice the value compared to baseline. Moreover, LoComp+ enhanced performance through the reorganized layout and there was an additional performance improvement of 15% compared to LoComp. Consequently, LoComp and LoComp+ minimized the overhead for the reliability of dense data structures in DRAMs without redundant ECC chips.

REFERENCES

[1] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. 41st Annu. Int. Symp. Comput. Archit.*, Minneapolis, MN, USA, 2014, pp. 361–372.
[2] T. J. Dell, *A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory*, IBM Microelectron. Div., New York, NY, USA, 1997.
[3] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Burlington, MA, USA: Morgan Kaufmann, 2010.
[4] *A Comparison of Client and Enterprise SSD Data Path Protection*. Accessed: Sep. 2016. [Online]. Available: https://www.micron.com
[5] *Intel Solid-State Drive DC S3500 Series*. Accessed: Oct. 2016. [Online]. Available: http://www.intel.com
[6] H. Zheng *et al.*, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *Proc. 41st IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Nov. 2008, pp. 210–221.
[7] D. H. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," in *Proc. 15th Ed. ASPLOS Archit. Support Program. Lang. Oper. Syst.*, Pittsburgh, Pennsylvania, USA, 2010, pp. 397–408.

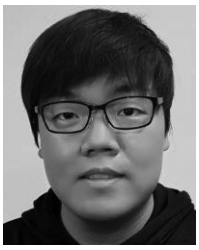
[8] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, Portland, OR, USA, 2012, pp. 285–296.
[9] L. Chen, Y. Cao, and Z. Zhang, "E3CC: A memory error protection scheme with novel address mapping for subranked and low-power memories," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 1–22, Dec. 2013.
[10] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *Proc. 31st Annu. Int. Symp. Comput. Archit.*, Munich, Germany, Jun. 2004, pp. 212–223.
[11] L. Chen, Y. Cao, and Z. Zhang, "Free ECC: An efficient error protection for compressed last-level caches," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Asheville, NC, USA, Oct. 2013, pp. 278–285.
[12] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring unconventional benefits from memory compression," in *Proc. IEEE 20th Int. Symp. High Perf. Comput. Archit. (HPCA)*, Orlando, FL, USA, Feb. 2014, pp. 638–649.
[13] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "COP: To compress and protect main memory," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Portland, OR, USA, Jun. 2015, pp. 682–693.
[14] J. Kim, M. Sullivan, S.-L. Gong, and M. Erez, "Frugal ECC: Efficient and versatile memory error protection through fine-grained compression," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–12.
[15] *Data Compression in the Intel® Solid-State Drive 520 Series*. Accessed: Sep. 2016. [Online]. Available: www.intel.com/go/ssd
[16] D. Kim *et al.*, "Exploiting compression-induced internal fragmentation for power-off recovery in SSD," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1720–1733, Jun. 2016.
[17] S. Lee *et al.*, "Warped-compression: Enabling power efficient GPUs through register compression," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Portland, OR, USA, 2015, pp. 502–514.
[18] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.*, Minneapolis, MN, USA, 2012, pp. 377–388.
[19] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.
[20] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan./Jun. 2011.
[21] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches," Dept. Comp. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Rep. 1500, 2004.
[22] S. Mittal and J. S. Vetter, "A survey of architectural approaches for data compression in cache and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1524–1536, May 2016.
[23] Q. S. Gao, "The Chinese remainder theorem and the prime memory system," in *Proc. 20th Annu. Int. Symp. Comput. Archit.*, San Diego, CA, USA, 1993, pp. 337–340.
[24] M. Gupta, J. S. Bhullar, and B. N. Bansal, "Undetected error probability of hamming code for any number of symbols," in *Proc. IEEE Int. Conf. Inf. Theory Inf. Security (ICITIS)*, Beijing, China, Dec. 2010, pp. 1015–1018.
[25] *PM863a Enterprise SSD Samsung V-NAND SSD*. Accessed: Aug. 2017. [Online]. Available: http://www.samsung.com
[26] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. 14th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Washington, DC, USA, 2009, pp. 229–240.
[27] *ECC Brings Reliability and Power Efficiency to Mobile Devices*. Accessed: Jul. 2017. [Online]. Available: https://www.micron.com
[28] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," *Commun. ACM*, vol. 54, no. 2, pp. 100–107, Feb. 2011.
[29] T. J. Dell, "System RAS implications of DRAM soft errors," *IBM J. Res. Develop.*, vol. 52, no. 3, pp. 307–314, May 2008.
[30] Y. Q. Shi, X. M. Zhang, Z.-C. Ni, and N. Ansari, "Interleaving for combating bursts of errors," *IEEE Circuits Syst. Mag.*, vol. 4, no. 1, pp. 29–42, Aug. 2004.
[31] *ARM Compiler Toolchain Compiler Reference*. Accessed: Jan. 2017. [Online]. Available: http://infocenter.arm.com/help/index.jsp
[32] N. D. Gulur, R. Manikantan, M. Mehendale, and R. Govindarajan, "Multiple sub-row buffers in DRAM: Unlocking performance and energy improvement opportunities," in *Proc. 26th ACM Int. Conf. Supercomput.*, Venice, Italy, 2012, pp. 257–266.
[33] T. Zhang *et al.*, "Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Minneapolis, MN, USA, Jun. 2014, pp. 349–360.
[34] M.-H. Teng, "Comments on 'the prime memory systems for array access,"' *IEEE Trans. Comput.*, vol. C-32, no. 11, p. 1072, Nov. 1983.
[35] *MT41j256m8 Data Sheet*. Accessed: Oct. 2016. [Online]. Available: http://www.micron.com/products/dram/
[36] *SPEC CPU 2006*. Accessed: Oct. 2016. [Online]. Available: http://www.spec.org/cpu2006
[37] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Jan. 2011. [Online]. Available: ftp://ftp.cs.princeton.edu/techreports/2010/890.pdf
[38] A. Gutierrez *et al.*, "Full-system analysis and characterization of interactive smartphone applications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Austin, TX, USA, Nov. 2011, pp. 81–90.
[39] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Architecting efficient interconnects for large caches with CACTI 6.0," *IEEE Micro*, vol. 28, no. 1, pp. 69–79, Jan./Feb. 2008.
[40] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts, "A fully integrated multi-CPU, GPU and memory controller 32nm processor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, USA, Feb. 2011, pp. 264–266.
[41] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, and B. Dhoedt, "Mobile device power models for energy efficient dynamic offloading at runtime," *J. Syst. Softw*, vol. 113, pp. 173–187, Mar. 2016.

Juhyung Hong received the B.S. and M.S. degrees in electrical and electronics engineering from Ajou University, Suwon, South Korea, in 2002 and 2004, respectively. He is currently pursuing the Ph.D. degree with Yonsei University, Seoul, South Korea.

He is a Senior Engineer with Samsung Electronics, Suwon. His current research interests include system on chip, NAND flash-based mass storage architecture, and memory system architecture.

Jeongbin Kim received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2010, where he is currently pursuing the Ph.D. degree.

His current research interests include future memory technology, NAND flash-based storage architecture, and system architecture.



Sangwoo Han received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree.

His current research interests include system-level architecture and design, and advanced storage systems and applications.

Eui-Young Chung (M'06) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2002.

From 1990 to 2005, he was a Principal Engineer with SoC Research and Development Center, Samsung Electronics, Yongin, South Korea. He is currently a Professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul. His current research interests include system architecture and very large scale integration design, including all aspects of computer-aided design with special emphasis on low-power applications, and flash memory applications.