2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

Attaché: Towards Ideal Memory Compression by Mitigating Metadata Bandwidth Overheads

Seokin Hong*†, Prashant J. Nair*
Bulent Abali, Alper Buyuktosunoglu, Kyu-Hyoun Kim, and Michael B. Healy
IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, USA
Email: [seokin.hong, prashant.nair.j]@ibm.com [abali, alperb, kimk, mbhealy]@us.ibm.com

***Abstract*—Memory systems are becoming bandwidth constrained and data compression is seen as a simple technique to increase their effective bandwidth. However, data compression requires accessing Metadata which incurs additional bandwidth overheads. Even after using a Metadata-Cache, the bandwidth overheads of Metadata can reduce the benefits of compression.**

This paper proposes *Attaché*, a framework that reduces the overheads of Metadata accesses. The Attaché framework consists of two components. The first component, called the *Blended Metadata* Engine (BLEM), enables data and its Metadata to be accessed together. BLEM incurs additional Metadata accesses only 0.003% times and removes almost all Metadata bandwidth overheads. The second component, called the *Compression Predictor* (COPR), predicts if the memory block is compressed. The COPR predictor uses a fine-grained line-level predictor, a coarse-grained page-level predictor, and a global indicator. This enables Attaché to predict the compressibility of the memory block before sending a memory read request. We implement Attaché on a memory system that uses Sub-Ranking. On average, Attaché achieves 15.3% speedup (ideal 17%) and saves 22% energy consumption (ideal 23%) when compared to a baseline system that does not employ data compression. Attaché is completely hardware-based and uses only 368KB of SRAM.

***Index Terms*—Data Compression, Metadata, Bandwidth, Sub-Ranking, Memory Systems**

I. Introduction

Increasing the main memory bandwidth is instrumental for increasing the performance of processor chips and accelerators [1], [2], [3], [4]. Designers have traditionally improved bandwidth by increasing the frequency or the pin count of the memory interface [5], [6], [7], [8], [9]. These techniques tend to have area and energy costs. Furthermore, interface changes are slow, as new memory standards are proposed only every 2 to 3 years [1], [2]. One can overcome these challenges by using data compression. Compression enables the memory systems to transfer data over fewer pins and fewer memory chips, thereby unlocking higher bandwidth [10]. However, identifying if the data is compressed requires additional Metadata [10], [11], [11], [12]. The additional memory accesses to Metadata can offset the benefits of compression. This paper tries to mitigate the memory access overheads of obtaining Metadata.

Main memory systems tend to comprise of Dynamic Random Access Memories (DRAM) [13], [14]. On a memory request, commodity memory modules transmit/receive 64 bytes of data (a cacheline). Each module typically contains 8 DRAM chips and each chip contributes 8 bytes of the 64-byte memory block. If we could compress a 64-byte block to a 32-byte block, then we can enable only 4 DRAM chips for each request, thereby doubling the effective bandwidth. This technique is called Sub-Ranking and together with data compression, Sub-Ranking can be used to improve memory bandwidth [15], [16], [17].

Unfortunately, the Metadata overheads involved in data compression can offset its bandwidth benefits. This is because each cacheline-size memory block from main memory will require a unique Metadata to identify its compressibility [10], [11], [12], [18]. For a high capacity memory system, it is impractical to store the Metadata in the memory controller. For instance, in a 16GB memory system if each cacheline requires 1 bit of Metadata, then the memory controller will need 32MB of storage. It is impractical to implement a 32MB SRAM in the memory controller due to latency, area, and energy overheads [19], [20], [21]. Therefore, Metadata tends to be stored in a separate location within the main memory and tends to require issuing an additional memory request.

One can reduce the bandwidth overhead of Metadata by employing a small Metadata-Cache within the memory controller [10]. Unfortunately, additional Metadata-Cache eviction and install requests can reduce the performance benefits of data compression. Figure 1 shows the proportion of compressed memory blocks (30 Bytes) and the additional bandwidth overheads of Metadata accesses. For this analysis, we use a reasonably large 1MB Metadata-Cache inside the Memory Controller. We also plot the values for SPEC [22] and GAP [23] workloads. Ideally, we can reduce the additional memory requests for accessing Metadata by up to 85%. The goal of this paper is to reduce almost all Metadata accesses. To this end, this paper proposes *Attaché*, a framework that helps mitigate Metadata accesses to provide a near-ideal speedup.

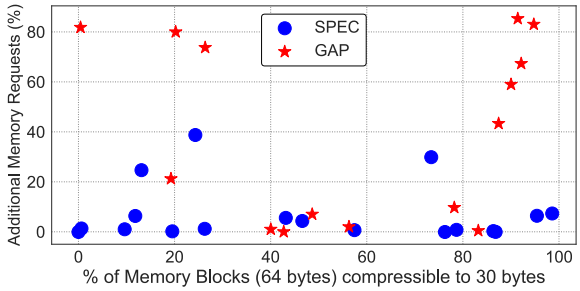


Fig. 1: The memory access overheads of Metadata accesses for SPEC [22] and GAP [23] benchmarks. Metadata can increase the memory traffic by up to 85%. The goal of this paper is to mitigate this overhead and obtain near-ideal speedup.

*Both authors contributed equally towards this paper.
†Currently affiliated with Kyungpook National University, South Korea. Email: seokin@knu.ac.kr

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00034

Attaché tackles Metadata bandwidth overheads by storing the Metadata of a cacheline-size memory block within the block itself. On a memory read, Attaché gets data and Metadata together in one access. Therefore, Attaché avoids almost all additional accesses to the memory. To this end, Attaché consists of two components. The first component, called the *Blended Metadata Engine* (BLEM), tries to place data and Metadata together. The second component, called *Compression Predictor* (COPR), predicts the values of Metadata for memory accesses. The predictor enables the memory controller to proactively issue memory requests only to the predicted Sub-Ranks.

The Blended Metadata Engine (BLEM) tries to place the data and Metadata in the same memory block during writes to memory. This is easy when the cacheline is compressible, as compression creates additional space for storing Metadata. However, if the cacheline is not compressed, then there is no additional space for Metadata. To overcome this problem, irrespective of whether the data is compressed or not, BLEM interprets the first few bits of the cacheline as its Metadata-header. A Metadata-header consists of a Compressed ID (CID) and an Exclusive ID (XID).

The Compressed ID (CID) identifies the compression status of the cacheline. The CID value is chosen randomly at boot-time and stored in the memory controller. For instance, if we have a 15-bit CID, the memory controller appends the 15-bit CID value to the compressed cacheline during a write. As each Sub-Rank stores 32 bytes of compressed data, for a 15-bit CID, the target compression size of a cacheline is 30 bytes. For a 64-byte cacheline that cannot be compressed to 30 bytes, the memory controller does not append the CID value (as there is no additional space) and simply writes-back the uncompressed data into both the Sub-Ranks (32 Bytes in each Sub-Rank).

During a read, if the line is compressed, then its top 15-bits will match the CID value. Therefore, the CID value can be used to identify compressed memory blocks. Unfortunately, while reading uncompressed 64-byte memory blocks, the top 15-bits can match the CID value by chance and there can be false-positives. In fact, for a 15-bit CID value, there is a $\frac{1}{2^{15}}$ chance (0.003%) that the top 15-bits of the uncompressed memory blocks will match the CID value [1]. This makes it difficult to identify whether the block is really compressed or simply a false positive. We refer to such cases as CID collisions and BLEM must identify all CID collisions.

The second part of Metadata-header, a 1-bit Exclusive ID (XID), helps identify CID collisions. XID is checked only if the top 15-bits of the data match the CID value. If the XID value is set to '1', it indicates a CID collision. During a write, if the memory block is compressible to 30 Bytes, then the XID value is reset to '0' and appended after the CID value. However, if the memory block is not compressible, then instead of writing-back the unmodified memory block, the memory controller checks the top 15-bits of the memory block for a CID collision. In case of a CID collision, the memory controller proactively writes an XID value of '1' as the 16^{th}-bit of the line, therefore altering the original data bit in the uncompressed memory block. The 16^{th} data-bit replaced by XID is written into a separate memory region called the Replacement Area. On a CID collision, the memory controller reads the entire 64-byte memory block and fetches the 16^{th} data-bit from the Replacement Area. BLEM incurs additional memory reads only during CID collisions. As CID collisions are rare (0.003% times), BLEM incurs negligible bandwidth overheads.

While BLEM minimizes the memory bandwidth overheads of Metadata to 0.003%, it only allows accessing Metadata **together with** data. However, to identify if both Sub-Ranks need to be enabled for a given memory block (when it is not compressed), the memory controller needs to access the Metadata **before** accessing data. To enable Metadata lookup before accessing data, several prior works have proposed using a Metadata-Cache. The Metadata-Cache stores the most recently accessed Metadata. Unfortunately, Metadata-Cache has bandwidth overheads as cache-misses have additional evictions and replacement memory requests.

This paper proposes replacing the Metadata-Cache with a predictor. This predictor, called a Compression Predictor (COPR), predicts the compression status of the memory block. The COPR consists of three levels of predictors. The first level, called the line-level predictor (LiPR), provides predictions for cachelines within a DRAM page. The second level, called the page-level predictor (PaPR), provides predictions for the compressibility of different DRAM pages. The third level, called the global indicator (GI), is a simple two-bit counter that keeps tracks of the compressibility of the most recent four memory accesses.

After a 32-byte memory block is read, the memory controller can check if its Metadata prediction was correct by interpreting the Metadata from BLEM. If the prediction was incorrect, the memory controller takes a corrective action by reading the remaining 32-byte memory block. As BLEM ensures that the Metadata is always read/written from/into the memory, COPR does not cause additional Metadata accesses.

Overall this paper has the following contributions:

1) **Blending Data and Metadata (BLEM)**, a technique to encode Metadata within the data irrespective of the compressibility of the data. Thereby eliminating almost all bandwidth overheads (99.997%) of Metadata accesses.
2) **A Simple Compression Predictor**, a high-confidence predictor that enables the BLEM engine to guess the compressibility of memory blocks.

Attaché provides 15.3% speedup and 22% energy reduction by efficiently compressing data into a system that uses Sub-Ranking. Attaché does not require any software support. Furthermore, Attaché requires only 368KB of SRAM for the predictor and a single register to store the CID value.

II. Background and Motivation

We provide a brief background on the main memory organization and data compression. Furthermore, we also highlight the bandwidth overheads involved in managing Metadata.

[1]Modern memory systems randomize data by performing data scrambling. Due to this, the probability of a 15-bit CID false positive for any uncompressed data memory block is 0.003%.

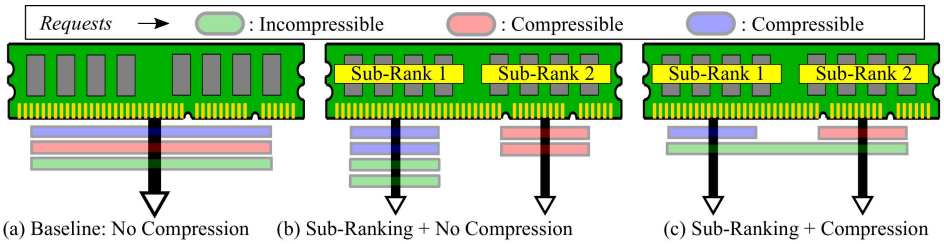


Fig. 2: Comparison of Baseline System, Sub-Ranking, and Sub-Ranking with compression. Figure (a) shows the baseline system which does not employ Sub-Ranking or compression. Figure (b) shows that Sub-Ranking can unlock 2x bandwidth but can also increase latency by 2x. Figure (c) shows that Sub-Ranking with compression can reduce the latency overheads of Sub-Ranking and still provide 2x bandwidth for compressible lines (more requests/second).

A. Main Memory System Organization

Commodity memory modules consist of several DRAM chips. Each DRAM chip consists of several bank-groups and each bank-group is further divided into multiple banks. Each bank contains millions of DRAM cells and operates independently within a channel [1], [2], [24], [25], [26].

A memory request from the memory controller activates a row of DRAM cells in the same bank across all chips. The row of activated DRAM cells is read into the row-buffer. As each memory module typically consists of 8 DRAM chips, the row buffer is split across each chip. For instance, in the case of an 8KB row buffer, each chip will hold 8Kb of data in its row buffer. Each memory request will fetch the appropriate 64 bits data from the 8Kb row buffer of a chip. As there are 8 chips, each memory request accesses 64 bytes of data [27].

B. Memory Bandwidth and Latency Tradeoff

As each memory module consists of several DRAM chips, one can improve bandwidth up to 2x allowing individual memory requests to be serviced by independently by 4 DRAM chips. However, each DRAM chip will now have to provide 128 bits of data as compared to 64 bits (2x more) and will increase memory access latency by 2x [27], [28] [2]. To reduce latency, commodity memory modules tend to operate all DRAM chips in lockstep as shown in Figure 2(a). This reduces the memory access latency while providing tolerable bandwidth [15], [30]. Ideally, we would like to get higher bandwidth with no additional latency overheads.

C. Get Bandwidth without Paying Latency

One can unlock higher bandwidth without incurring additional latency by using Sub-Ranking with Compression.

1) Sub-Ranking to Unlock Higher Bandwidth: One way to improve bandwidth is by enabling the memory request to access a smaller group of DRAM chips within a module. Each group of DRAM chips is called a Sub-Rank [15]. Unfortunately, simply Sub-Ranking memory modules increase its latency [16], [17]. For instance, if a memory module has two Sub-Ranks, then each memory request is catered by 4 DRAM chips instead of 8 DRAM chips. This means that each chip will still have to provide 2x more data and this will increase the access latency by 2x as shown in Figure 2(b).

[2]One may also use pseudo-channels in HBM2 [29] without additional latency overheads.

2) Compress Data to Reduce Latency: One way to mitigate the additional access latency from Sub-Ranking is by compressing data. For instance, if a 64-byte memory block is compressed to at least 32 Bytes, then each Sub-Rank can provide this compressed data with 2x lower access latency (same latency as the baseline system) as shown in Figure 2(c). Therefore, Sub-Ranking with compression is a practical low-latency solution for enabling high bandwidth memory systems [15], [31].

D. Main Memory Data Compression

To compress data, memory systems can use low-latency algorithms that are implemented in the memory controller.

1) Efficient Data Compression Algorithms: Data values tend to be similar within a cacheline. The Base-Delta-Immediate (BDI) algorithm uses this insight to compress cacheline-size data blocks [12], [18]. Similarly, the Frequent-Pattern-Compression (FPC) [32] algorithm keeps track of frequently occurring data patterns. Data patterns like all-zeros tend to occur frequently and FPC represents these patterns with fewer bits. To this end, FPC requires only a small lookup table.

2) Compression-Decompression Engine: The data compression is performed by a compression-decompression engine in the memory controller as shown in Figure 3. The compression-decompression engine typically runs one or more compression algorithms. During a write, the compression engine is activated and the data is compressed as it flows through the engine into the Sub-Rank. During a read, the decompression engine is activated and the data is decompressed as it flows from the Sub-Rank to the cores and caches.

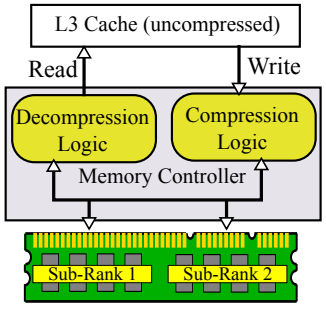


Fig. 3: The compression-decompression engine for a Sub-Ranked memory system (not drawn to scale). Reads flow through the decompression engine and writes flow through the compression engine.

A key metric for a compression-decompression engine is its latency. Fortunately, BDI requires only simple arithmetic operations and FPC requires only table lookups. These compression algorithms can compress-decompress within 1 cycle.

E. Data Compression: Potential

To understand the effectiveness of data compression, we look at the compressibility of cachelines (64-byte memory blocks) for memory intensive SPEC and GAP workloads. Figure 4 shows the percentage of cachelines that can be compressed to less than 30 bytes.

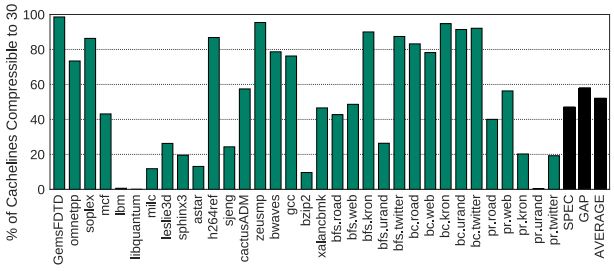


Fig. 4: The percentage of cachelines compressible to 30 Bytes. On average, 50% of the cachelines (64-byte memory blocks) are compressible to 30 Bytes and can be stored within a Sub-Rank without additional latency overheads.

On average, 50% of the cachelines are compressible to 30 bytes. For these compressible lines, we can potentially obtain 2x higher bandwidth by using Sub-Ranking. For the remaining lines, we can disable Sub-Ranking and still maintain 1x bandwidth. Therefore, on average, we can ideally obtain 1.5x higher bandwidth by employing data compression with Sub-Ranking. However, in practice, a memory system employing compression accesses additional Metadata and this lowers the bandwidth benefits of compression.

F. Data Compression: Metadata Overheads

A memory system that employs compression also maintains additional Metadata information for each memory block. For instance, the decompression engine will require a 1-bit per memory block to identify its compression status.

1) Capacity Overheads: Even if each 64-byte memory block has only 1 bit of Metadata, the main memory contains millions of memory blocks. Therefore, the overall capacity overheads of Metadata tends to be large. For instance, a 16GB main memory system will need 32MB of Metadata. Due to its large size, it is impractical to place Metadata on the processor chip. Therefore, Metadata is typically stored in a separate region within the main memory with a capacity overhead of only 0.2%.

2) Bandwidth Overheads: Storing Metadata in main memory has bandwidth overheads. Each memory request requires an additional access to the Metadata region. These additional accesses tend to lower the benefits of data compression.

G. Pitfalls of using a Metadata-Cache

A Metadata-Cache can be used to reduce the memory bandwidth overheads of Metadata. If the Metadata is present within the Metadata-Cache (cache hit), then no additional memory accesses are required. However, if Metadata is absent within the Metadata-Cache (cache miss), then two additional memory requests (eviction and install) may be necessary.

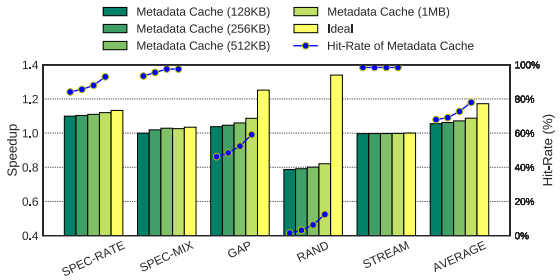


Fig. 5: The Impact of Metadata-Cache hit-rate on performance. On average, even after using a 1MB Metadata-Cache with a 77% hit-rate, we can obtain only 8% speedup.

We can increase the hit-rate of the Metadata-Cache by increasing its size. Unfortunately, it is impractical to create a large Metadata-Cache within the memory controller. Furthermore, a large Metadata-Cache will incur a significant lookup latency and also increases the chip area considerably. Figure 5 shows the average hit-rate of different sizes of Metadata-Cache. On average, even with an impractically large 1MB Metadata-Cache, we can obtain only 8% speedup [33].

III. Attaché: An Overview

Figure 6 shows an overview of Attaché for a 16GB main memory system. Broadly, Attaché consists of two components. The first component of Attaché, called the Blended Metadata Engine (BLEM) embeds Metadata with data for both compressed and uncompressed cachelines. To this end, BLEM interprets the first few bits of data as the Metadata header that consists of Compression ID (CID) and Exclusive ID (XID). For cachelines that are uncompressed and their CID value collides, BLEM will replace a data-bit with the XID value of 1. BLEM also uses a Replacement Area (RA) to store bits that are replaced by XID. The second component of Attaché, called the Compression Predictor (COPR), helps predict if a cacheline is compressible or not. COPR acts as a replacement to the Metadata-Cache and avoids all bandwidth overheads that result from managing the Metadata-Cache. Attaché uses a compression-decompression engine to compress/decompress memory blocks and provide the compressibility information of the memory blocks to BLEM and COPR.

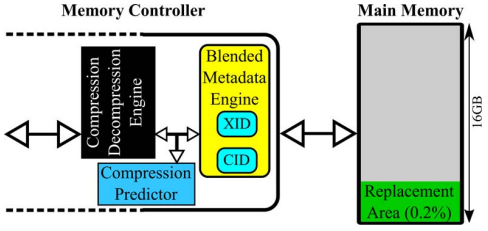


Fig. 6: An Overview of Attaché (not drawn to scale). Attaché consists of a Blended Metadata Engine (BLEM) and a Compression Predictor (COPR). The compression-decompression engine will compress and decompress memory blocks. The Replacement Area (0.2% of main memory) stores the bits that are replaced by XID.

IV. The Attaché Framework

This section describes the design of Attaché framework. The Attaché framework broadly consists of the Blended Metadata Engine (BLEM) and the Compression Predictor Unit (COPR).

A. The Blended Metadata Engine (BLEM)

The Blended Metadata Engine (BLEM) aims to store Metadata and data together for all cachelines, irrespective of their compressibility. However, traditional memory systems face challenges in storing and accessing Metadata.

1) Pitfalls of Conventional Metadata Storage: Prior work on memory compression places data and its Metadata in the same row-buffer [10]. The memory controller can fetch the data and Metadata by issuing two consecutive read requests to the same row-buffer. This reduces the latency of fetching Metadata. Figure 7 shows such an implementation. Typically a DRAM row-buffer is 8KB (or 128 cachelines) and each Metadata memory request accesses 64 Bytes (or 512 bits) of Metadata. Therefore, even if a data cacheline uses 4-bits of Metadata, a Metadata memory request can prefetch Metadata for 127 data cachelines in the DRAM row.

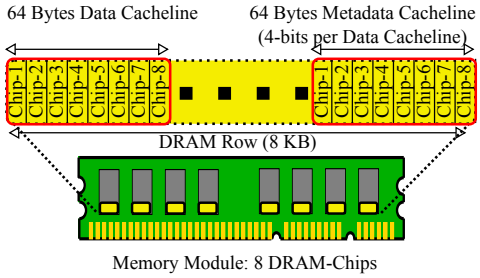


Fig. 7: The conventional technique for storing Metadata. Metadata is stored in the same row-buffer as data. Therefore, Metadata and data can be accessed consecutively using two memory requests to the same row-buffer.

Storing Metadata in this manner has three drawbacks. First, Metadata will need to be accessed before accessing data. This is because the memory controller will need the Metadata to figure out the Sub-Rank or Sub-Ranks (if data is uncompressed) that need to be enabled. This increases the data access latency and read traffic from main memory. Second, for several workloads, only a few cachelines tend to be accessed within a row-buffer. Therefore, fetching Metadata for all cachelines in a row-buffer is wasteful. Thirdly, the newly fetched Metadata cacheline can cause the eviction of another cacheline from the Metadata-cache. This increases the write traffic to the main memory. If we can somehow fetch Metadata in the same access as the data, then we can eliminate the drawbacks of storing Metadata in the memory.

2) Insight: Intelligently prepend Metadata with data: One can try to prepend Metadata with data. This is simple for compressible cachelines as compressing a cacheline creates additional space within the cacheline for storing Metadata. However, for cachelines that are not compressed, there is no additional space to store the Metadata.

Attaché stores Metadata by interpreting the first few bits of the cacheline as the Metadata-Header and comparing it against a Metadata-header that is stored in the memory controller. The Attaché framework performs this comparison irrespective of the compressibility of the cacheline. The Metadata-Header has two components; the Compression ID and the Exclusive ID.

3) Identify Compression with Compression ID (CID): Attaché uses the Compression ID (CID) to identify if the cacheline is compressed or not. During writes, the CID is prepended in front of the compressed cachelines. For cachelines that are not compressed, there is no additional space for storing CID. Therefore, the memory controller simply writes the lines as they are. During a read, the memory controller uses the insight that lines that do not have CID as the first few bits are uncompressed lines. However, as there are millions of lines in the memory system, it is possible that the first few bits of some uncompressed lines will match the CID values. We refer to this scenario of false positives as a CID collision.

4) Probability of CID Collision and the size of CID: The probability of CID collision depends on the length of CID. For instance, if CID is 3-bits long, then CID will collide every 8 memory requests (12.5% probability). In this paper, we use a 15-bit CID to reduce the probability of CID collision to 0.003% (i.e $\frac{1}{2^{15}}$). Figure 8 shows the probability of CID collision with the number of accesses to uncompressed lines. Even in the worst case, if all accesses are only to uncompressed lines, a 15-bit CID collides only every 32K accesses.

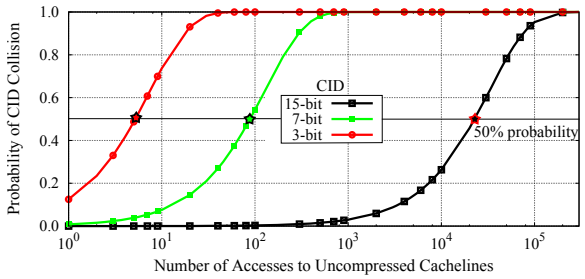


Fig. 8: The probability of a CID collision versus the number of accesses to uncompressed lines. For a 15-bit CID, we expect a CID collision every 32K accesses.

5) Extending CID to store more information: Currently, by using only a 15-bit CID, Attaché can identify if the cacheline is compressible or not. However, it is possible that the cacheline is dynamically compressed by choosing either the BDI or the FPC algorithm. To decompress such a cacheline, we will need to add more Metadata information. To this end, we simply reduce the size of CID to be 14-bits and use the 15th bit to identify the compression algorithm. CID can be easily extended to store additional information. Table I shows the CID size, the number of information bits and the probability of collision.

TABLE I: Extending CID to store additional information

CID Size	Additional Information Bits	Probability of Collision
15	0	0.003%
14	1	0.006%
13	2	0.01 %

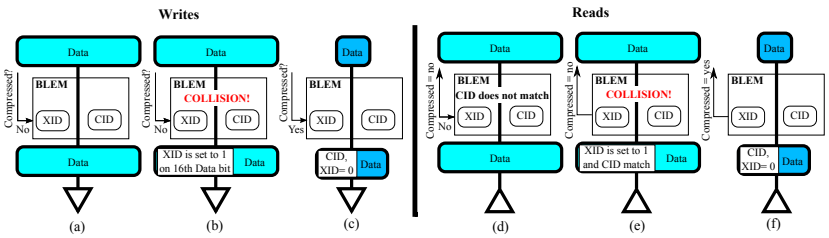


Fig. 9: Analysis of BLEM for reads and writes. During writes, BLEM proactively inserts XID = 1 only for uncompressed lines that have CID collisions. During reads, BLEM uses this information to identify collisions.

Using this insight, CID can be extended to cater to multiple compression algorithms on the fly. Unfortunately, even a large CID will collide at some point in time. If a CID collision is not identified, the memory controller will wrongly interpret the uncompressed line as a compressed line and cause data corruption. Therefore, CID collisions will need to be detected.

6) Detecting CID collisions with Exclusive ID (XID): A 1-bit Exclusive ID (XID) is used to identify CID collisions. During writes, if there is a CID collision, the memory controller proactively replaces the 16^{th} data-bit as XID equal to '1'. It stores the replaced data-bit in a separate area called the Replacement Area (RA). If there is no collision, the memory controller does not modify the uncompressed line. Therefore, even if all memory blocks are uncompressed, the Replacement Area will incur accesses only 0.003% of the time. If the memory blocks are compressed, then the memory controller will simply append an XID equal to '0' after the CID. During reads, the Replacement Area is read-only if the CID matches and the 16^{th} data-bit is set to '1' (collision).

7) Handling the data-bit replaced By XID: To cater to the worst case in which all cachelines can encounter a CID collision, each cacheline in the memory system has 1-bit in the Replacement Area and indexes into it in a direct-mapped manner. The area overhead of the Replacement Area is only 0.2% (i.e. $\frac{1}{512}$) of the total capacity of the main memory. The Replacement Area is invisible to the OS and is visible only to the memory controller.

B. BLEM: Implementation and Flow

For reliability and security, memory controllers scramble memory blocks using a Scrambling-Descrambling unit [34], [35]. After scrambling, the data value appears pseudo-random. Interpreting the Metadata-Header in the BLEM engine after the data flows through the Scrambling unit will ensure that the expected 15-bit CID collision probability is 0.003% [3].

Figure 9 shows how BLEM works for reads and writes. BLEM uses the compression information from the Compression-Decompression engine and can disable the Compression-Decompression engine for uncompressed lines [4]. As shown in Figure 9(a), on a write request, if data cannot be compressed, BLEM writes data as it is. However, as shown in Figure 9(b), on a CID collision, BLEM proactively inserts XID as '1' in the 16^{th} data-bit. Thereafter, it will write the 16^{th} data-bit into the Replacement Area (not shown in the Figure). As shown in Figure 9(c), for compressed lines, BLEM simply prepends the CID and XID of '0' in front of the data.

As shown in Figure 9(d), on a read request, if the top 15-bits do not match the CID, BLEM simply reads the data and as the data is deemed not compressed. However, as shown in Figure 9(e), if CID matches and XID is set to '1' (collision), then BLEM fetches the 16^{th} data-bit from the Replacement Area (not shown in the Figure) and does not decompress the data. As shown in Figure 9(f), for compressed lines, BLEM simply checks if the CID matches and XID is set to '0' and sends these lines to be decompressed.

C. Compression Predictor (COPR): Design

While BLEM reduces the Metadata bandwidth overheads, it delivers Metadata together with data. However, to identify if Sub-Ranking needs to be enabled during reads, the memory controller will need to access Metadata before accessing data. Therefore, prior works have proposed using a Metadata-Cache.

1) Pitfalls of Metadata Caches: Unlike regular caches, Metadata-Caches tend to reside within the memory controller. The goal of the Metadata-Cache is to keep the most recently accessed Metadata. Furthermore, the Metadata-Cache must be amenable to a small lookup latency by the memory controller. This is important as the memory controller will probe the Metadata-Cache for every memory request and if the lookup latency is large, it can cause slowdown. Therefore, prior work in both industry and academia have assumed Metadata-Caches that are typically small [10], [33]. In this paper, we optimistically assume a 1MB Metadata-Cache (impractical) within the memory controller.

Our analysis shows that a 1MB Metadata-Cache achieves a hit-rate of 77%. Unfortunately, additional memory accesses are involved for the 23% of the requests that miss on the Metadata-Cache. In case of a Metadata-Cache miss, the memory controller needs to issue a separate request to the Metadata region in the DRAM row to fetch the new Metadata. After the new Metadata is fetched, the Metadata-Cache should evict a line of Metadata to install the new Metadata line. If the evicted line is dirty, then the Metadata-Cache will require the memory controller to issue another write request.

2) Compression Predictor to mitigate overheads: The key drawback of the Metadata-Cache is that it is in charge of managing Metadata. Therefore, Metadata-Cache must ensure that Metadata is always kept updated in the main memory and incurs bandwidth overheads. However, in the case of

[3]Scrambling-Descrambling units tend to choose hashes with memory block address as an input. This ensures that even if the same data is written in every memory block, data will still appear pseudo-random [36].

[4]As Scrambling-Descrambling unit is placed after the Compression-Decompression engine, scrambling the data has no effect on its compressibility.

Attaché, BLEM is responsible for managing the Metadata. Due to this, the Attaché framework does not need a Metadata-Cache for issuing write-back and install requests to or from the main memory. Due to this, the Attaché framework can replace the Metadata-Cache with a predictor called the Compression Predictor (COPR).

On a read, the COPR simply predicts the compression status of the cacheline. After reading the cacheline and passing it through BLEM, the memory controller can determine if the COPR predictor was correct. Thereafter, the memory controller updates COPR with the correct prediction. If COPR can obtain the atleast the same prediction accuracy as the hit-rate of the Metadata-Cache, then COPR will mitigate the additional memory accesses that stem from the Metadata-Cache.

3) COPR: Enabling multi-granularity prediction: The memory system can have compressible cachelines that are sparsely distributed over an entire row or evenly spread throughout the row. For instance, there may be rows that have a few cachelines that are compressible to 30B. On the other hand, there may also be rows in which all cachelines are compressible to 30B. To predict both these instances, COPR employs a multi-granularity predictor. The multi-granularity predictor is composed of three components as shown in Figure 10.

The Global Indicator (GI): GI tries to provide an overall information of the data-compressibility of the running applications. GI is composed of *eight two-bit saturating counters*, each of which keeps track of the compressibility of $\frac{1}{8}^{th}$ the memory space. At boot time, the counter values are initialized to zero. For every memory request, a counter corresponding to the requested memory space is incremented by one when the cacheline is compressible, otherwise it is reinitialized to zero. GI can be used as an accurate indicator for predicting the compressibility within a memory space if there is abundant similarity in compressibility.

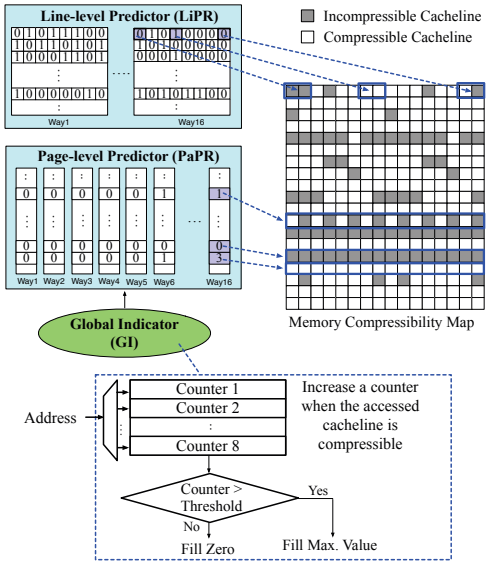


Fig. 10: The Compression Predictor (COPR). COPR contains three components; the Global Indicator (GI), the Page-level Predictor(PaPR), and the Line-Level Predictor (LiPR).

Page-Level Predictor (PaPR): By exploiting the similarity in the compressibility of cachelines within an OS page [12], [18], [37], PaPR provides compression predictions at the page granularity. PaPR is a small set-associative cache structure that is indexed by the page number. Each entry of PaPR is a two-bit saturating counter. On memory access, if the accessed cacheline is compressible, then the PaPR entry indexed by the page number is incremented. In case the cacheline is incompressible, the entry is decremented. If the entry value is greater than or equal to 2, then the cachelines of the corresponding page are predicted to be compressible. Otherwise, cachelines are predicted to be incompressible. If there is no entry for the accessed page, PaPR allocates a new entry for the page with an initial value determined by the GI. When the corresponding counter value of GI is greater than a threshold value, the new entry is set to its maximum value (i.e., 3), otherwise it is reset to zero. For this paper, we use PaPR of 192KB[5].

Line-Level Predictor (LiPR): The goal of LiPR is to provide compression predictions at 64-byte cacheline (memory block) granularity. LiPR helps predict the compressibility of cachelines even for pages with low similarity in compressibility of cachelines. LiPR is a set-associative cache structure indexed by the page number. Each LiPR entry is 64 bits long and represents the compressibility of contiguous cachelines in a page. After a cacheline indexes into LiPR entry, LiPR updates the entry value if the prediction is not correct. At this time, LiPR also proactively updates the neighboring entries to the current entry. To this end, LiPR uses the two-bit values of PaPR to determine if the neighboring cachelines have the same compressibility. If the two-bit values are greater than or equal to 2, LiPR updates the neighboring entries because the page is deemed to have similar compressibility across cachelines. If the two-bit values are less than 2, then the page has a mix of compressible and incompressible cachelines. For this paper, we use LiPR of 176KB[5].

4) Prediction accuracy of COPR: Figure 11 shows the prediction accuracy of COPR. On average, COPR provides a prediction accuracy of 88% which is 8% higher than the hit-rate of a 1MB Metadata-Cache. COPR achieves this by using the insight that neighboring lines have similar compressibility and that certain rows tend to have all lines that are compressible. Furthermore, even for benchmarks that produce a low prediction accuracy, BLEM does not require any Metadata accesses.

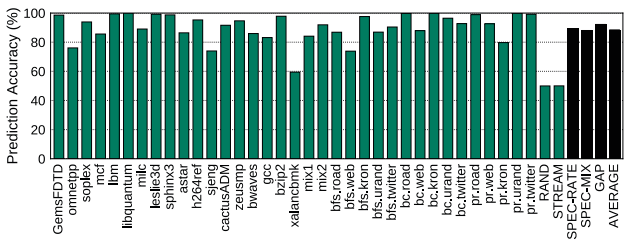


Fig. 11: The prediction accuracy of Compression Predictor (COPR). On average, COPR provides an accuracy of 88%.

[5]The area overhead of COPR (192KB+176KB), which is estimated using McPAT, is only 0.9% of the chip area in the baseline [38].

D. Change in Compressibility and Software Support

Attaché does not use the free space made available by compression. Therefore, as far as the capacity goes, the real to physical (compressed) memory sizes are the same. Thus, even if the compressibility of data changes, the data overflowing the memory capacity is not possible in Attaché.

Attache is a completely hardware-based approach. The main memory provisions 0.2% of the capacity for a reserved area and does not expose this area to the OS. The memory modules simply show a reduced capacity (99.8%) to the OS at boot time. Therefore Attache requires no software interactions and does not change the Virtual to Physical memory management. The memory controller alone handles the compression-decompression and metadata management.

E. Tying it together: Attaché with Sub-Ranking

To highlight the benefits of Attaché, this paper evaluates Attaché on a memory system that uses Sub-Ranking. It should be noted that the implementation of the Attaché framework is not limited to a system that uses Sub-Ranking and the Attaché framework can be easily applied to other proposals as well.

In memory system that uses two Sub-Ranks, we can unlock 2x the bandwidth for cachelines that are compressed to 30 bytes (32 bytes with Metadata-header). We can disable Sub-Ranking for uncompressed cachelines and maintain the same latency as the baseline. For simplicity, our implementation tries to compress cachelines in odd-numbered rows into the first Sub-Rank and cachelines in even-numbered rows into the second Sub-Rank.

If the memory controller receives a read request to an odd-numbered row, Attaché checks COPR to predict if the cacheline is compressed or not. If COPR says that the cacheline is compressed, then it fetches the cacheline entirely from the first Sub-Rank with the same latency as the baseline. After reading the line, BLEM can interpret the Metadata and determine if the prediction was accurate. If the prediction is inaccurate, then Attaché simply fetches the rest of the cacheline by enabling the second Sub-Rank. If COPR predicts that the cacheline is not compressed, then both the Sub-Ranks are enabled and the entire 64 Byte cacheline is fetched. The same process is followed for read requests to even-numbered row. However, the top 32 Bytes of the uncompressed data are flipped and stored. Therefore, in the case of uncompressed cachelines, the memory controller will fetch the top 32 Bytes from the second Sub-Rank. This is important as the Metadata-Header is stored in the top 2 Bytes by design. After the read operation, Attaché updates COPR with the correct values of Metadata.

V. Experimental Methodology

To evaluate the performance and energy benefits of Attaché, we use the SST simulation framework [39]. For modeling the processor core, we use the *Ariel* core component in SST. Ariel is a simple core model that dynamically generates memory instruction streams from a running application by using a Pintool. We extended the Ariel core to model detailed out-of-order (OoO) execution.

TABLE II: Baseline System Configuration

Number of cores (OoO)	8
Processor clock speed	4 GHz
Issue width	4
Last Level Cache (Shared)	8MB, 8-Way, 64-byte lines
LLC access latency	20 cycles
Memory bus speed	1600 MHz
Memory channels	2
Ranks per channel	1
Banks Groups	4
Banks per Bank Group	4
Rows per bank	64K
Memory blocks (64 bytes) per row	128
DRAM Access Timings: T_{RCD}-T_{RP}-T_{CAS}	22-22-22
DRAM Refresh Timings: T_{RFC}/T_{REFI}	350ns/7.8s

To model the memory system, we use *CramSim* [40]. CramSim is a cycle-accurate main memory component of SST. CramSim enforces strict timing and also models JEDEC DDR4 protocol specifications. Each rank in the memory module has 8 DRAM chips and can have two sub-ranks. This is enabled by provisioning two different chip-select signals for groups of 4 DRAM chips. We configure CramSim to prioritize read requests over write requests. The memory controller also has a write buffer that drains writes to the main memory once a high watermark is reached. To enable compression, the compression engine compresses a memory block using both BDI and FPC, and selects the one with the best compression ratio for the block. As per prior work, we assume that compression and decompression of data occurs within 1 cycle [11]. For both Metadata-Cache and COPR, we assume that the access latency is 8 cycles, which is the same as that of the L2 cache. CramSim models the energy and power overheads using a DRAMSim2 style power calculator [41]. The baseline system does not employ compression and Sub-Ranking. The parameters for the baseline system are shown in Table II.

For our evaluations, we chose memory intensive benchmarks, which have greater than "1 Miss Per 1000 Instructions" from Last Level Cache, from the SPEC2006 [22] and GAP suites [23]. We warm up the caches and the memory for 40 Billion instructions and execute 4 Billion instructions. We execute all benchmarks in the rate mode, in which all eight cores execute the same benchmark. We also evaluate two 8-threaded mixed workloads, which are created by classifying workloads and forming four categories from highly compressible to incompressible. We randomly pick two benchmarks from each category to form mixed workloads.

VI. Results

This section discusses the performance and energy benefits of Attaché, and the sensitivity of Attaché to design parameters.

A. Performance

Figure 12 shows the performance gains of using Attaché when compared to a system that only uses Metadata Caching. On average, Attaché provides 15.3% speedup across all benchmarks using only 368KB of COPR [6]. This is almost close to the ideal case of 17%. On the other hand, a system that uses

[6]As the data is scrambled, irrespective of the benchmark and its data contents, only 0.003% of the memory accesses go into the replacement area, on average. For instance, even if the baseline system had data that were all 0s, after data scrambling, the data written to the memory tends to be random 64 Byte strings. The negligible bandwidth overhead into the replacement area tends to be imperceivable in performance and energy simulations.

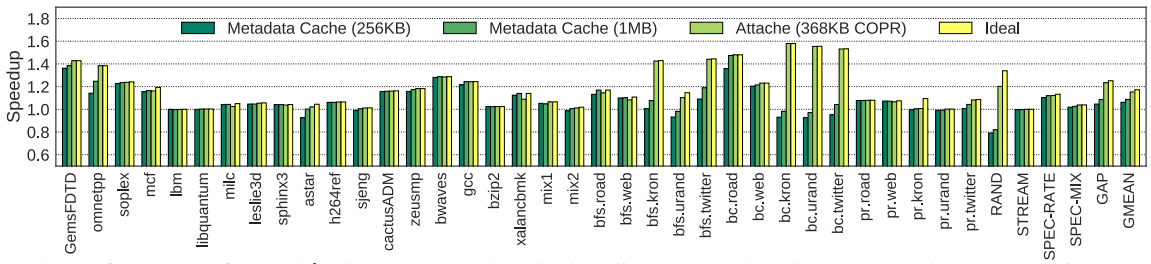


Fig. 12: The performance of Attaché when compared to the baseline system that does not employ compression. On average, Attaché provides 15.3% speedup (ideal 17%) which is 7% higher than a system that uses Metadata Caching. The results on synthetic RAND and STREAM benchmarks highlight the robustness of Attaché to regular and irregular data patterns.

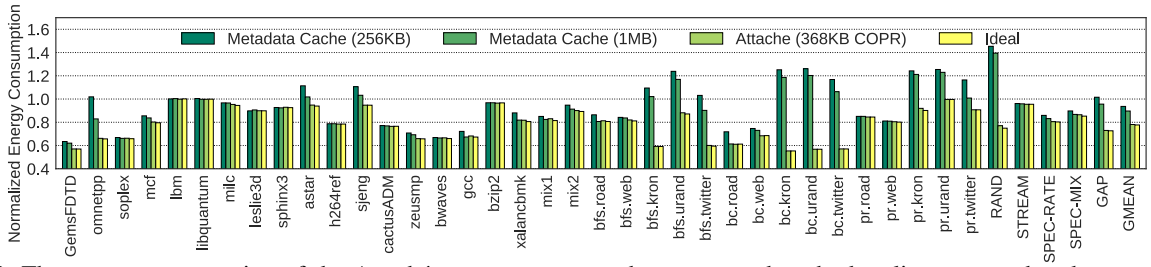
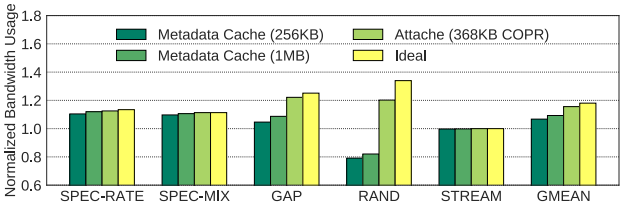


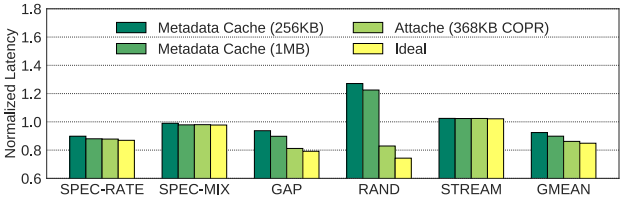
Fig. 13: The energy consumption of the Attaché memory system when compared to the baseline system that does not employ compression. On average, across High-MPKI SPEC benchmarks, Attaché reduces energy consumption by 22% (ideal 23%) which is 12% higher than a system that uses Metadata Caching. The results on synthetic RAND and STREAM benchmarks highlight the energy benefits of Attaché irrespective of the data patterns.

a 1MB Metadata Caching only provides 8% speedup. Attaché remains robust over synthetic benchmarks and consistently provides speedup irrespective of the access pattern. On the other hand, Metadata-Cache is not useful for the *RAND* benchmark and therefore it shows a slowdown of 17%.

The performance gains of using Attaché come from its bandwidth improvement. Figure 14(a) shows the memory bandwidth usage and Figure 14(b) shows the average memory latency. On average, Attaché enables 16% higher bandwidth which results in 14% lower average memory latency.



(a) Memory Bandwidth Usage



(b) Average Memory Latency

Fig. 14: Memory bandwidth improvement and latency reduction due to Attaché.

The performance of Metadata caching is correlated to the Metadata-Cache hit rate. For instance, *bc.kron* shows a slowdown as it has a poor hit-rate in the Metadata Cache. Benchmarks like *libquantum* are not compressible, and as it is a streaming application, there is no performance gain for it. Additionally, while a system with Metadata caches will require additional write-back operations, Attaché does not require this for any benchmark. Due to this, even for the *RAND* benchmark, Attaché has no additional bandwidth overheads from Metadata.

B. Energy Reduction

Figure 13 shows the energy benefits of using Attaché when compared to a system that only uses Metadata Caching. On average, Attaché saves 22% energy across all benchmarks, close to the ideal case of 23%. On the other hand, a system that uses Metadata caching only provides energy savings of 10%. The energy savings of Attaché also sustain over synthetic benchmarks. On the other hand, Metadata-cache is not useful for the *RAND* benchmark and therefore shows an increased energy consumption of 40%.

The energy overhead of Metadata Caching is correlated to the Metadata-Cache miss-rate. For instance, *bc.kron* shows 20% higher energy as it has a low hit-rate in the Metadata Cache. For the same benchmark, Attaché shows a lower energy consumption as it does not incur any additional Metadata accesses. Benchmarks like *libquantum* are not compressible, however as it is a streaming application, there are no energy savings for it. Attaché consistently saves energy when compared to the baseline for all benchmarks.

C. Additional traffic due to Metadata-cache

Figure 15 shows the additional memory traffic due to Metadata caching. On average, even a large Metadata-cache increases the memory traffic by 25%. This is because of cache evictions and installs from the Metadata region in the main memory. Furthermore, most additional requests are read

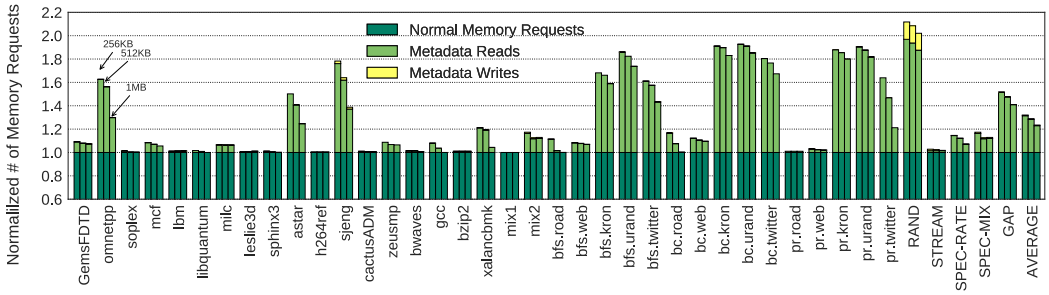


Fig. 15: Normalized number of memory requests in a system that uses Metadata caching. On average, even a large 1MB Metadata-cache incurs 25% additional memory accesses due to evictions and installs.

requests (installs). This is because the compressibility of the line does not change much during its lifetime. Therefore, the Metadata bits associated with the compressibility of cachelines tends to remain the same. As a result, Metadata cachelines tend to be predominantly clean.

D. Sensitivity to Replacement Algorithm

Figure 16 shows the hit-rates of a 1MB Metadata cache with state-of-the-art replacement policies like DRRIP and SHIP [42], [43]. As compared to last level caches that have hit rates between 40% to 60%, Metadata caches tend to have a much higher hit-rate (77%) [42], [43]. Therefore, other state-of-art policies like DRRIP and SHIP provide only a 2% increase in the hit-rate of the Metadata cache.

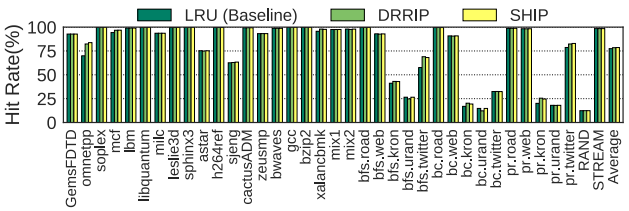


Fig. 16: Hit rate of a 1MB Metadata-Cache with different replacement policies. The baseline LRU policy provides a very high hit-rate (77%) and other policies only increase the hit-rate by 2%.

E. Sensitivity to Predictor Components

Different components of COPR contribute to its performance benefits. Figure 17 shows that PaPR (page-level predictor) alone can provide 11.5% speedup. However, after combining with GI (global indicator) can provide 15.3% speedup. The LiPR (line-level predictor) is only useful for mixed workloads.

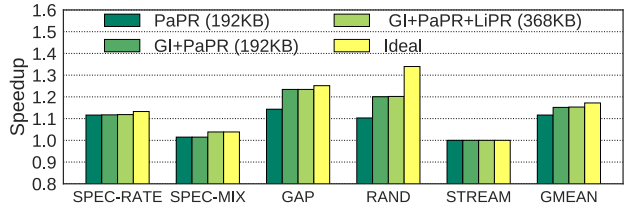


Fig. 17: The performance with different components of COPR. On average, except for mixed benchmarks, the PaPR (page-level predictor) and GI (global indicator) provide most of the prediction accuracy.

VII. Related Work

In this section, we describe prior work that is closely related to our work.

A. Data Compression for Main Memory

Memzip [10] compresses data for improving the bandwidth of the main memory. To this end, Memzip employs static Sub-Ranking and uses a Metadata-Cache. Attaché can be easily applied to the Memzip framework to reduce metadata overheads. A prior work by Lee and Hong used the XRL compression algorithm for low latency decompression [44]. While lower compression-decompression latency improves performance, it does not reduce the overheads of Metadata caching. Pekhimenko et. al. [12] proposed an efficient technique for the main memory compression. However, this compression technique is focused on improving the memory capacity. Similarly, Abali et. al [45] proposed an alternative technique of implementing compression in main memories, primarily for improving memory capacity. These techniques for compression do not focus on primarily mitigating the bandwidth overheads of Metadata. Non-Volatile Memories can also use compression to reduce energy and improve performance [46].

Deb et. al. [21] describes the challenges in maintaining Metadata and recommend using ECC storage to store Metadata with a predictor. While this technique is useful for memory modules that have ECC in them, commodity memory modules do not have ECC chips [34], [47], [48], [49]. Attaché is applicable to non-ECC systems as well. Furthermore, the predictor by Deb et. al. [21] is different from COPR. COPR employs a multi-granularity predictor which enables COPR to track compression status at page-level and line-level granularity.

Compression is also useful for increasing the effective memory capacity. Prior works from the industry such as IBM MXT and VMWare ESX use “Balloon Drivers” to allocate and hold unused memory when data becomes incompressible or when Virtual Machines exceed capacity thresholds [50], [51], [52], [53], [54]. Similarly, Kim et. al. proposed “Dual Memory Compression” that helps improve memory capacity. Attaché is a bandwidth optimization technique and does not improve memory capacity [55]. Nevertheless, Attaché can be used for efficient metadata storage in those systems too.

B. Data Compression for Energy Savings

Kim et al. [56] introduce a bit-plane compression technique that uses a bit-plane transformation to achieve a high compres-

sion ratio. They exploit this high compression ratio to save bandwidth. Pekhimenko et. al. [57] addresses an inefficiency of the memory compression techniques and its consequent increase in bus energy consumption. This is because compression naturally increases the number of bit toggles. These work are efficient at improving the memory interface performance and energy consumption. Unfortunately, these techniques do not address the Metadata bandwidth overheads and therefore Attaché can be easily applied to these techniques.

C. Data Compression for Caches

Prior work has also focussed on using compression to improve the effective cache-capacity and lower energy [32], [58]. To this end, prior work focuses on the similarity of data content present in caches and employ compression. In the similar spirit, Young et. al. [11] use compression in DRAM caches to improve both capacity and bandwidth dynamically. While this work tries to improve the bandwidth of DRAM caches, it does not incur any Metadata overheads. This is because the tag storage in caches acts as natural avenues for storing Metadata and can be fetched together with data. Attaché works well for systems, such as main memories, that do not have this design and tend to be bandwidth constrained.

Caches can also use a dictionary-based compression scheme for higher compressibility [59]. Attaché can be easily used with schemes for obtaining Metadata along with data. Other cache-compression techniques like YCC, SCC, Sc2, and Decoupled Compressed Cache can easily use Attaché to fetch data and Metadata together [60], [61], [62], [63].

D. Compression for NOCs

Thuresson et al. [64] try to improve the bandwidth of the network on chips by compressing cachelines. In this work, the compression is primarily employed between the LLC and the memory controller. Providing additional lanes for Metadata at this interface is relatively simpler. This is because the NOC is designed by the processor vendor and therefore can be customized. However, the DDR interface is a standard and therefore it is challenging to transmit Metadata over the memory interface without any additional accesses. Nevertheless, Attaché can easily be extended to other levels of the memory hierarchy.

E. Data Compression for GPUs

Sathish et al. [65] try to save memory bandwidth for GPUs. In this work, they propose using both lossy and lossless compression for GPU content. Attaché is orthogonal to the compression technology and characteristics. Therefore, Attaché can easily be applied over this work for GPUs to get additional bandwidth savings.

F. Other Relevant Work

Sardashti and Wood [37] observe that cachelines in the same page may not have similar compressibility. If the compressibility of a page varies, the Metadata cache will still perform poorer than COPR. Due to varying compressibility, the traffic from Metadata cache will now dominate write-backs.

Hallnor et. al. [66] proposed using compressed data throughout the memory hierarchy. Such an approach reduces the overheads of compression and decompression at every level in the hierarchy. Attaché is orthogonal to this approach and can be used with this approach at every level to improve bandwidth efficiency even further.

Recent work by Han et. al. [67] and Kadetotad et. al. [68] used compression with deep neural networks to significantly improve performance and reduce energy. This work primarily focused on improving capacity. However, even to improve bandwidth, frameworks like Attaché will help such deep networks as it can save Metadata bandwidth for every level of the deep neural network.

New technologies like stacked memories and non-volatile memories tend to have ECC, security primitives, and endurance optimizations for increasing their lifetime [69], [70], [71], [72], [73]. Attaché can be applied independently to any of these schemes and is broadly applicable. Going forward, Attaché can also be co-optimized with caching policies [42], [43] and memory-request scheduling policies [74].

VIII. Summary

Main memory systems are bandwidth constrained and data compression is a practical technique to improve their bandwidth. While one can use Sub-Ranking to unlock higher bandwidth, one cannot simply employ data compression to utilize this bandwidth. This is because data compression relies on Metadata for identifying the compressibility of the cacheline. Therefore each access to the line will also require an additional access to its metadata. While one can resolve this issue by completely placing the Metadata in an on-chip buffer near the processor, such a design will not be scalable. For instance, in a 16GB memory system, if each cacheline requires 1 bit of Metadata, the size of the buffer will be 32MB. Therefore, prior work has proposed caching Metadata in an on-chip Metadata-cache. This paper observes that even at high Metadata-cache hit-rates Metadata-Cache requests can cause bandwidth overheads.

To this end, this paper proposes Attaché, a technique that reduces the bandwidth overheads of Metadata accesses to 0.003%. Attaché is a scalable technique and can be implemented without any software changes. For a 16GB main memory, the hardware overhead of Attaché is only 368KB. Furthermore, Attaché requires 0.2% Replacement Area in the main memory for storing data from cachelines that incur CID collisions. Overall, Attaché provides 15.3% higher performance (ideal is 17%) and 22% lower energy (ideal is 23%) by reducing the Metadata bandwidth for compression.

Acknowledgments

We thank the anonymous reviewers for their feedback. We also thank Paul Crumley, Ravi Nair, Pradip Bose, and Hillery Hunter for their discussions on data compression and memory systems. This project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B621073.

REFERENCES

[1] JEDEC Standard, "DDR3 Standard," in *JESD79-3E*, 2015.

[2] JEDEC Standard, "DDR4 Standard," in *JESD79-4*, 2015.

[3] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, May 2011.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[5] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, Feb 2014.

[6] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun, "A 1.2v 12.8gb/s 2gb mobile wide-i/o dram with 4x128 i/os using tsv-based stacking," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, Feb 2011.

[7] JEDEC Standard, "High Bandwidth Memory (HBM) DRAM," in *JESD235*, 2013.

[8] H. M. C. Consortium, "Hybrid memory cube specification 1.0," 2013. [Online]. Available: http://www.hybridmemorycube.org

[9] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016.

[10] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "Memzip: Exploring unconventional benefits from memory compression," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014.

[11] V. Young, P. J. Nair, and M. K. Qureshi, "Dice: Compressing dram caches for bandwidth and capacity," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080243

[12] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540724

[13] B. L. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[14] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485929

[15] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000100

[16] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, Nov 2008.

[17] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore dimm: an energy efficient memory module with independently controlled drams," *IEEE Computer Architecture Letters*, vol. 8, Jan 2009.

[18] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept 2012.

[19] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob, "Technology comparison for large last-level caches (l3cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013.

[20] D. Roberts, N. Kim, and T. Mudge, "On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology," in *Digital System Design Architectures, Methods and Tools, pp. 570-578, Aug. 2007*.

[21] A. Deb, P. Faraboschi, A. Shafiee, N. Muralimanohar, R. Balasubramonian, and R. Schreiber, "Enabling technologies for memory compression: Metadata, mapping, and prediction," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016.

[22] The SPEC2006 Benchmark Suite, "www.spec.org."

[23] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[24] P. J. Nair, C.-C. Chou, and M. K. Qureshi, "Refresh pausing in dram memory systems," *ACM Trans. Archit. Code Optim.*, vol. 11, Feb. 2014. [Online]. Available: http://doi.acm.org/10.1145/2579669

[25] P. Nair, C.-C. Chou, and M. Qureshi, "A case for refresh pausing in dram memory systems," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, Feb 2013.

[26] M. K. Qureshi, D. Kim, S. Khan, P. J. Nair, and O. Mutlu, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015.

[27] *JESD79-3F.pdf*, JEDEC, July 2012.

[28] *ddr3_8gb_1.5v_twindie_x4x8.pdf - Rev. C 4/13 EN*, Micron, 2011.

[29] H. Jun, "Hbm (high bandwidth memory) for 2.5d," in *Semicon Taiwan, year=2015*.

[30] D. H. Yoon, M. K. Jeong, M. Sullivan, and M. Erez, "The dynamic granularity memory system," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337159.2337222

[31] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez, "Balancing dram locality and parallelism in shared memory cmp systems," in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, ser. HPCA '12. Washington, DC, USA: IEEE Computer Society, 2012. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2012.6168944

[32] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[33] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, 2016.

[34] P. J. Nair, V. Sridharan, and M. K. Qureshi, "Xed: Exposing on-die error detection information for strong memory reliability," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[35] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018.

[36] D. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE Computer Architecture Letters*, vol. 14, Jan 2015.

[37] S. Sardashti and D. A. Wood, "Could compression be of general use? evaluating memory compression across domains," *ACM Trans. Archit. Code Optim.*, vol. 14, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3138805

[38] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd An-*

nual IEEE/ACM International Symposium on Microarchitecture (MICRO), Dec 2009.

[39] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[40] M. B. Healy and S. Hong, "Cramsim: Controller and memory simulator," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3132402.3132408

[41] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "Usimm: the utah simulated memory module a simulation infrastructure for the jwac memory scheduling championship," 2012.

[42] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815971

[43] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, Jr., and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155671

[44] J.-S. Lee, W.-K. Hong, and S.-D. Kim, "A selective compressed memory system by on-line data decompressing," in *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, vol. 1, 1999.

[45] B. Abali, H. Franke, X. Shen, D. E. Poff, and T. B. Smith, "Performance of hardware compressed main memory," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001.

[46] P. M. Palangappa and K. Mohanram, "Compex: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvm," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016.

[47] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Faultsim: A fast, configurable memory-reliability simulator for conventional and 3d-stacked systems," *ACM Trans. Archit. Code Optim.*, vol. 12, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2831234

[48] D. Roberts and P. Nair, "Faultsim: A fast, configurable memory-resilience simulator," in *The Memory Forum: In conjunction with ISCA*, vol. 41, 2014.

[49] C. Chou, P. Nair, and M. K. Qureshi, "Reducing refresh power in mobile devices with morphable ecc," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015.

[50] P. Franaszek and J. Robinson, "Design and analysis of internal organizations for compressed random access memories," in *IBM Report, RC 21146, year=1998*.

[51] C. D. Benveniste, P. A. Franaszek, and J. T. Robinson, "Cache-memory interfaces in compressed memory systems," *IEEE Transactions on Computers*, vol. 50, Nov 2001.

[52] T. B. Smith, B. Abali, D. E. Poff, and R. B. Tremaine, "Memory expansion technology (mxt): Competitive impact," *IBM Journal of Research and Development*, vol. 45, March 2001.

[53] R. B. Tremaine, T. B. Smith, M. Wazlowski, D. Har, K.-K. Mak, and S. Arramreddy, "Pinnacle: Ibm mxt in a memory controller chip," *IEEE Micro*, vol. 21, Mar 2001.

[54] E. VMware, "Understanding memory resource management in vmware esx 4.1."

[55] S. Kim, S. Lee, T. Kim, and J. Huh, "Transparent dual memory compression architecture," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept 2017.

[56] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[57] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "A case for toggle-aware compression for gpu systems," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016.

[58] S. Kim, J. Lee, J. Kim, and S. Hong, "Residue cache: A low-energy low-area l2 cache architecture via compression and partial hits," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155670

[59] B. Panda and A. Seznec, "Dictionary sharing: An efficient cache compression scheme for compressed caches," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016.

[60] S. Sardashti, A. Seznec, and D. A. Wood, "Yet another compressed cache: A low-cost yet effective compressed cache," *ACM Trans. Archit. Code Optim.*, vol. 13, Sep. 2016. [Online]. Available: http://doi.acm.org/10.1145/2976740

[61] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014.

[62] A. Arelakis and P. Stenstrom, "Sc2: A statistical compression cache scheme," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665696

[63] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy optimization," *IEEE Micro*, vol. 34, May 2014.

[64] M. Thuresson, L. Spracklen, and P. Stenstrom, "Memory-link compression schemes: A value locality perspective," *IEEE Transactions on Computers*, vol. 57, July 2008.

[65] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory i/o link compression for improving performance of gpgpu workloads," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370864

[66] E. G. Hallnor and S. K. Reinhardt, "A unified compressed memory hierarchy," in *11th International Symposium on High-Performance Computer Architecture*, Feb 2005.

[67] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[68] D. Kadetotad, S. Arunachalam, C. Chakrabarti, and J.-s. Seo, "Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2966986.2967028

[69] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Citadel: Efficiently protecting stacked memory from large granularity failures," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014.

[70] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Citadel: Efficiently protecting stacked memory from tsv and large granularity failures," *ACM Trans. Archit. Code Optim.*, vol. 12, Jan. 2016. [Online]. Available: http://doi.acm.org/10.1145/2840807

[71] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Cop: To compress and protect main memory," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015.

[72] P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi, "Reducing read latency of phase change memory via early read and turbo read," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015.

[73] V. Young, P. J. Nair, and M. K. Qureshi, "Deuce: Write-efficient encryption for non-volatile memories," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2694344.2694387

[74] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, Jan 2010.