Main Memory Compression

Yunhua Fang

November 2024

1 Introduction

Modern computing systems are increasingly strained by data-intensive applications, which demand both high memory capacity and bandwidth. With the exponential growth of tasks in artificial intelligence, real-time data analytics, high-resolution graphics, and large-scale simulations, memory systems are being pushed to their limits. Main memory, typically composed of DRAM, is critical in these systems as it acts as an intermediate storage layer that feeds data to high-speed processors. However, scaling memory to meet the needs of these applications poses several significant challenges. As applications grow in complexity and data requirements, maintaining sufficient memory capacity and bandwidth to prevent performance degradation is both costly and power-intensive.

Take mainstream tasks as examples. State-of-the-art models in artificial intelligence, especially large language models (LLMs) and complex neural networks, have escalated memory and bandwidth demands to unprecedented levels. For instance, the size and compute needs of LLMs have been growing exponentially, with model parameter sizes increasing by a factor of 410x and compute needs by 750x every two years from 2018 to 2022 [1]. Figure 1 presents a clear trends. However, this rapid scaling has outpaced advances in DRAM and interconnect bandwidth, which have only grown by 1.6x and 1.4x, respectively, over the same period, creating a widening gap known as the "Memory Wall".

The "Memory Wall" represents a critical bottleneck in modern computing systems, where advancements in processor speeds far outpace improvements in main memory latency and bandwidth. As processor capabilities have increased exponentially, DRAM technology has lagged significantly behind. While the compute power of processors has improved by roughly 80% per year, DRAM speed only increases by about 7% per year, creating a widening performance gap where memory, rather than computation, becomes the limiting factor for system performance [3]. This disparity means that even with high cache hit rates, the growing latency of DRAM accesses leads to increasingly frequent slowdowns in data retrieval, stalling otherwise efficient processors [3].

Capacity constraints add another layer to the problem. The working sets of modern applications often exceed the physical capacity of main memory, leading to reliance on slower storage options such as SSDs or hard drives, which further degrades performance. Expanding DRAM capacity is not always feasible due to cost and power limitations, especially in energy-sensitive environments like data centers and mobile devices. Additionally, increasing physical memory alone cannot address the limitations imposed by the fixed data bus sizes and latency-sensitive applications that cannot afford delays in data retrieval.

The result is that while computing power increases, memory and communication limitations become the bottleneck, especially during tasks that require intensive data transfer, like transformer inference. This disparity restricts the effective deployment and scaling of state-of-the-art models on current hardware. Current main memory, predominantly DRAM, is also constrained by power consumption and physical limitations, making it infeasible to simply expand capacity to meet these growing needs. Increasing DRAM capacity raises costs and energy consumption, which is not sustainable for many applications, particularly in data centers or mobile devices where efficiency is paramount. Thus, overcoming the memory wall and addressing these limitations require innovative memory management techniques to reduce the volume of data and increase the effective bandwidth. Such solutions are essential to enable the continued scaling of State-of-the-art models and maintain performance in the face of growing memory demands.

Main memory compression offers a promising approach to these challenges by reducing the memory footprint of data stored in DRAM. By storing data in a compressed format, memory compression effectively increases the usable memory capacity and reduces data transfer times, as smaller data packets require less bandwidth. For example, studies show that memory compression techniques can increase effective memory capacity by up to 69% while reducing bandwidth consumption by as much as 24%, significantly alleviating memory-related bottlenecks [4]. Furthermore, by reducing memory access requirements, compression also lowers energy consumption, providing a dual benefit for both performance and power efficiency.

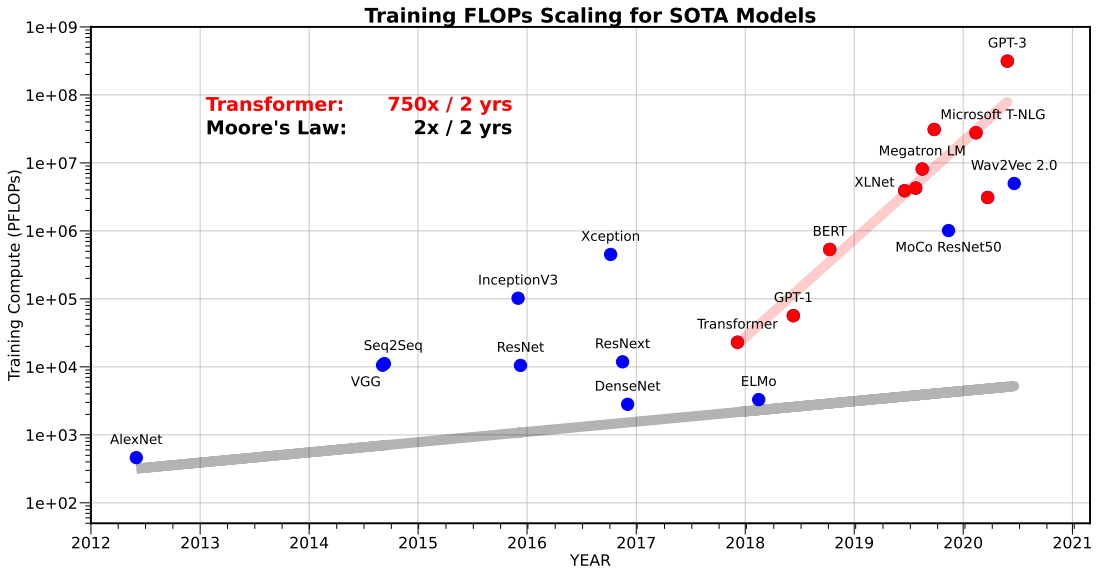

Figure 1: The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different CV, NLP, and Speech models, along with the different scaling of Transformer models (750x/2yrs) [1]

This report delves into the principles and impact of main memory compression, examining its role in addressing capacity, bandwidth, and energy efficiency limitations. By analyzing advanced memory compression techniques, we aim to demonstrate how memory compression can extend the scalability of modern computing systems, alleviate the memory wall bottleneck, and sustain the performance requirements of emerging state-of-the-art models. Through this exploration, we will underscore the critical importance of memory compression in overcoming current limitations and paving the way for continued innovation in high-performance computing.

2 Methods

Non-volatile memory technologies such as STT-MRAM [5] address the refresh energy overheads of DRAM by eliminating the need for periodic refresh operations. However, the Memory Wall persists, limiting system performance despite these energy savings. To overcome this challenge and better utilize CPU capabilities, modern computer systems implement main memory compression, an indirect approach to enhancing bandwidth. During the compression process, metadata is generated to facilitate the restoration of the original data. The effectiveness of main memory compression depends significantly on efficient metadata management, as poorly handled metadata can introduce additional overhead and reduce system performance.

To address this, two mainstream approaches have emerged for integrating metadata into main memory compression frameworks: utilizing a metadata cache and compressing metadata alongside data blocks. Each approach offers distinct trade-offs in terms of complexity, efficiency, and performance. In the following sections, we explore advanced techniques for main memory compression with metadata and evaluate their performance in modern computer systems, focusing on their ability to mitigate the Memory Wall and maximize resource utilization.

2.1 Metadata Cache

A metadata cache is a specialized hardware component used in main memory compression systems to store frequently accessed metadata efficiently. Metadata is essential for translating memory addresses in compressed memory, as it provides information about the location, size, and state of compressed data blocks. Without a metadata cache, each memory access would require fetching metadata from the main memory, adding significant latency to every operation. By caching this metadata close to the processor, the system reduces the need for repeated memory accesses, improving overall efficiency.

For example, the Linearly Compressed Pages (LCP) framework introduces a fixed-size compression approach where each cache line within a compressed memory page is reduced to the same size [4]. This linear compression simplifies address translation and metadata management, as the positions of compressed cache lines can be determined using simple arithmetic operations rather than complex lookups. LCP divides each memory page into three regions: compressed data, metadata, and exceptions. Metadata provides essential information about compressed cache lines, while exceptions handle uncompressible or overflow data. The framework also incorporates a metadata cache to store recently accessed metadata, minimizing the latency associated with metadata retrieval and ensuring fast address translation. The LCP enhances memory

capacity by efficiently compressing data, achieving an average increase of 69% in effective capacity while reducing bandwidth usage by 24%.

Although the metadata cache plays a crucial role in bridging the performance gap introduced by compression overheads, enabling the system to leverage the benefits of main memory compression—such as increased capacity and bandwidth—while keeping latency and energy consumption low, its efficiency depends heavily on its design and the workload characteristics. Inappropriate design brings additional bandwidth and lookup overheads to access the metadata cache in a different storage space. The main memory compression framework should make it an essential focus in the development.

Previous hardware main memory compression such as LCP [4], RMC [8], and DMC [9] requires additional metadata access and Operating System support if data is incompressible. Those methods are lack of adaptivity. A new pragmatic hardware main memory compression architecture, Compresso, is designed to enhance memory capacity and bandwidth while minimizing the performance overhead associated with traditional memory compression techniques [6]. Compresso focuses on reducing the additional data movement and latency overheads traditionally associated with memory compression. It employs a modified Bit-Plane Compression (BPC) algorithm [9], optimized for both compression ratio and energy efficiency, which operates at a granularity of 64 bytes to match CPU cache line sizes. A key feature of Compresso is its LinePack packing mechanism, which efficiently organizes compressed cache lines to minimize fragmentation and improve memory utilization. Furthermore, Compresso incorporates several data movement optimizations, such as alignment-friendly cache line sizes, dynamic inflation room expansion, and page overflow prediction, to handle variable data sizes while maintaining high performance. By maintaining metadata in a dedicated memory space and leveraging a specialized metadata cache, Compresso minimizes the latency and overhead associated with compressed memory address translation.

The benefits of Compresso are significant. First, it achieves a 1.85x increase in memory capacity on average, allowing more data to be stored in the same physical memory space. This reduces the need for frequent secondary storage access, lowering energy consumption and improving tail latency. Second, Compresso demonstrates a 24% speedup over competitive compressed systems in single-core environments and a 27% speedup in multi-core systems, highlighting its ability to enhance performance while maintaining OS transparency. Finally, the architecture supports a wide range of applications without requiring software or OS modifications, simplifying deployment and ensuring compatibility with modern systems.

The metadata cache improves the performance of main memory compression by ensuring that most metadata lookups occur at the cache level rather than in slower main memory. This reduces the additional overhead caused by compression, such as address translation and managing variable-sized compressed data.

2.2 Metadata in Compressed Data Blocks

However, the metadata cache is not without drawbacks. The reliance on a metadata cache means that workloads with high metadata miss rates, such as certain graph processing benchmarks, may experience degraded performance. Figure 2 shows the memory access overheads of Metadata accesses for SPEC [10] and GAP [11] benchmarks. Metadata can increase the memory traffic by up to 85%. Studies show that even with an impractically large 1MB metadata cache achieving a 77% hit rate, the additional memory traffic still increases by up to 25% due to evictions and installs [12]. Additionally, while its various optimizations reduce data movement overheads, these mechanisms introduce complexity in hardware design, potentially increasing development and manufacturing costs.

Another type of framework mitigates these issues by embedding metadata directly within the compressed data block, removing the need for a separate metadata cache. By storing metadata alongside the data, it eliminates the bandwidth and latency overheads associated with metadata cache management. This approach also simplifies the memory controller design by reducing the dependency on complex cache replacement policies and metadata synchronization mechanisms. Furthermore, eliminating the metadata cache reduces chip area requirements and energy consumption, as metadata caching often incurs significant energy overheads during lookups and updates

Attache is an innovative memory compression framework designed to address the performance bottlenecks and inefficiencies caused by traditional metadata management in compressed memory systems [12]. The motivation behind Attache stems from the significant overhead introduced by metadata caches, which, while mitigating metadata retrieval latency, generate additional memory traffic due to cache misses and evictions. This problem is exacerbated in multi-core systems and memory-intensive workloads, where frequent metadata lookups and cache contention can negate the benefits of compression. Attache reimagines metadata handling to minimize bandwidth overheads and improve the overall efficiency of compressed memory systems.

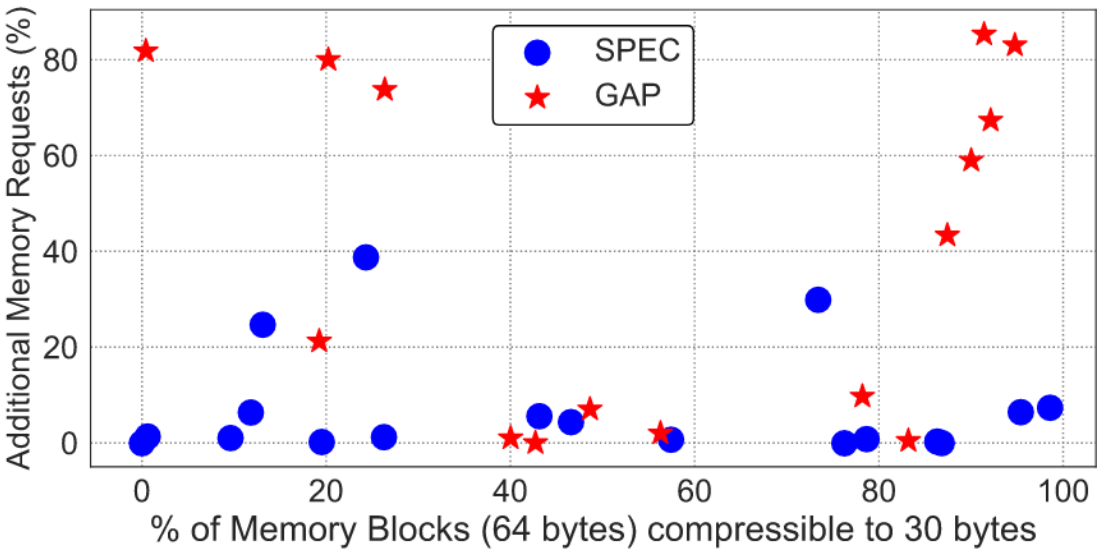

Figure 2: The memory access overheads of Metadata accesses for SPEC [10] and GAP [11] benchmarks. Metadata can increase the memory traffic by up to 85%.

The core methodology of Attache is its Blended Metadata Engine (BLEM), which integrates metadata directly within the compressed data block. By embedding metadata alongside the data it describes, Attache eliminates the need for a separate metadata cache, reducing memory traffic and latency associated with metadata management. BLEM operates by aligning metadata with data blocks at a granularity of cache lines, enabling efficient address translation and access with minimal overhead. Attache also incorporates a spatial indexing scheme, allowing the memory controller to quickly locate embedded metadata while preserving the simplicity of address translation logic. This approach ensures that metadata is always accessible without additional memory lookups or cache management, even in multi-core environments.

Approximate Memory Compression is a novel approach with a simple design concept by leveraging the error resilience of emerging workloads, such as machine learning and data analytics [13]. Programmers identify approximation-tolerant memory regions using a simple interface, while a runtime Quality Control Framework (QCF) modulates error constraints to ensure that the compressed data remains within the application's quality requirements. By reducing the precision, the data block size inherently decreases which expands the bandwidth. The compression scheme employs a bidirectional precision scaling approach, which truncates the least significant bits (LSBs) and most significant bits (MSBs) of data elements. The compression parameters, such as the number of truncated bits and padding values, are stored as metadata in the compressed block header, enabling efficient decompression.

Compression with Reduced Metadata Access (CRAM) also embeds metadata as a marker word within the data itself [14]. The marker indicates the compression status and location of data within the memory, effectively bypassing the need for explicit metadata lookups in a separate cache or memory region. To enhance access efficiency, CRAM incorporates a Line Location Predictor (LLP), a lightweight hardware module that predicts the location of compressed lines based on historical compressibility patterns and access behaviors, achieving up to 98% accuracy. Additionally, CRAM uses a marker collision resolution mechanism, such as data inversion, to handle cases where the marker word coincides with actual data, ensuring the reliability of metadata interpretation. These features work together to reduce bandwidth overhead, as memory accesses for metadata retrieval are minimized, and the prediction mechanism accelerates data access. Furthermore, CRAM dynamically decides whether to compress data based on its compressibility, applying compression only when it leads to tangible performance gains. This adaptive approach ensures that the framework maintains high efficiency across a variety of workloads without incurring unnecessary overhead.

3 Discussion

The choice of metadata management strategy is critical for achieving efficient main memory compression. Two dominant approaches—using a metadata cache and embedding metadata within compressed data blocks—offer distinct trade-offs in terms of performance, complexity, and scalability. This discussion explores the benefits and drawbacks of each architecture in detail, providing a comparative analysis of their effectiveness in modern computing systems.

A metadata cache provides a dedicated hardware resource for storing frequently accessed metadata, significantly reducing the latency of metadata lookups

during memory access. By caching metadata close to the processor, this approach minimizes the need for additional memory accesses, thereby improving performance and reducing the bandwidth overhead of managing compressed memory. The metadata cache design ensures compatibility with existing memory architectures and does not require significant software modifications, making it suitable for diverse applications.

Despite its advantages, the metadata cache approach has limitations, particularly in scenarios with high metadata miss rates. Cache misses necessitate additional memory accesses to retrieve metadata from main memory, increasing memory traffic and latency. For example, workloads with irregular access patterns, such as graph processing, may suffer from degraded performance due to frequent metadata cache misses. Moreover, the eviction and replacement policies required for metadata caches introduce further complexity and overhead. Additionally, the metadata cache requires significant chip area and incurs energy costs during lookups and updates, which can affect the scalability of this approach, especially in multi-core environments.

On the other hand, embedding metadata directly within compressed data blocks eliminates the need for a separate metadata cache, addressing many of the drawbacks associated with cache management. Attache's Blended Metadata Engine (BLEM) integrates metadata alongside data blocks, ensuring metadata is always accessible without additional memory lookups [12]. This reduces memory traffic and latency while simplifying the hardware design, as there is no need for complex cache replacement policies or synchronization mechanisms. Similarly, CRAM's implicit metadata approach leverages marker words within data blocks, avoiding the bandwidth overhead of separate metadata management and enabling efficient access prediction through mechanisms like the Line Location Predictor (LLP) [14]. These methods significantly improve energy efficiency and reduce the complexity of the memory controller design.

However, embedding metadata within data blocks introduces its own challenges. The primary concern is the potential reduction in overall compression ratios, as metadata occupies space within the compressed data region. This trade-off becomes particularly pronounced in workloads with high compressibility, where the inclusion of metadata may offset the benefits of compression. Additionally, mechanisms for resolving marker word collisions, such as data inversion in CRAM, add a layer of complexity to hardware implementation. While the embedded metadata approach avoids cache-related overheads, its reliance on spatial indexing and alignment introduces challenges for workloads with irregular or dynamic access patterns.

The metadata cache approach excels in scenarios with predictable and compressible workloads, where high metadata hit rates can be maintained, minimizing memory traffic and latency. It is particularly effective in systems where compatibility with existing architectures is critical. However, the embedded metadata strategy is better suited for environments with high variability and irregular access patterns, as it eliminates cache management overheads and ensures consistent access times. While embedding metadata simplifies the overall architecture, it may sacrifice compression efficiency and introduce implementation complexities, particularly in resolving marker conflicts and ensuring data alignment.

4 Conclusion

Both metadata cache and embedded metadata approaches provide valuable solutions to the challenges of metadata management in main memory compression. The choice of strategy depends on workload characteristics, system requirements, and implementation constraints. Future research could focus on hybrid designs that combine the strengths of both approaches, leveraging metadata caching for high hit-rate scenarios while embedding metadata for irregular workloads, thereby achieving a balance between performance, efficiency, and scalability.

5 Reference

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," IEEE Micro, vol. 44, no. 3, pp. 33–39, May 2024, doi: 10.1109/MM.2024.3373763.

[2] D. Patel, J. Koch, and W. Chu, "The Memory Wall: Past, Present, and Future of DRAM".

[3] Wm. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," SIGARCH Comput. Archit. News, vol. 23, no. 1, pp. 20–24, Mar. 1995, doi: 10.1145/216585.216588.

[4] G. Pekhimenko et al., "Linearly compressed pages: a low-complexity, low-latency main memory compression framework," in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, Davis California: ACM, Dec. 2013, pp. 172–184. doi: 10.1145/2540708.2540724.

[5] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS), Apr. 2013, pp. 256–267.

[6] E. Choukse, M. Erez, and A. R. Alameldeen, "Compresso: Pragmatic Main Memory Compression," in

2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka: IEEE, Oct. 2018, pp. 546–558. doi: 10.1109/MICRO.2018.00051.

[7] M. Ekman and P. Stenstrom, "A Robust Main-Memory Compression Scheme," in Proceedings of the 32nd Annual International Symposium on Computer Architecture, 2005, pp. 74–85.

[8] S. Kim, S. Lee, T. Kim, and J. Huh, "Transparent dual memory compression architecture," in Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques, 2017, pp. 206–218.

[9] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures," in Proceedings of the 43rd Annual International Symposium on Computer Architecture, 2016, pp. 329–340.

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

[11] S. Beamer, K. Asanovi´c, and D. Patterson, "The gap benchmark suite," arXiv preprint arXiv:1508.03619, 2015.

[12] S. Hong, P. J. Nair, B. Abali, A. Buyuktosunoglu, K.-H. Kim, and M. Healy, "Attaché: Towards Ideal Memory Compression by Mitigating Metadata Bandwidth Overheads," in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka: IEEE, Oct. 2018, pp. 326–338. doi: 10.1109/MICRO.2018.00034.

[13] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate Memory Compression," IEEE Trans. VLSI Syst., vol. 28, no. 4, pp. 980–991, Apr. 2020, doi: 10.1109/TVLSI.2020.2970041.

[14] V. Young, S. Kariyappa, and M. K. Qureshi, "CRAM: Efficient Hardware-Based Memory Compression for Bandwidth Enhancement," Jul. 19, 2018, arXiv: arXiv:1807.07685. Accessed: Sep. 10, 2024. [Online]. Available: http://arxiv.org/abs/1807.07685