Thesis for The Degree of Licentiate of Engineering

Techniques to Expand Capacity in Memory Hierarchies using Compression

Qi Shao

Division of Computer and Network Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Techniques to Expand Capacity in Memory Hierarchies using Compression

Qi Shao

Department of Computer Science & Engineering
Division of Computer and Network Systems
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2025.

"Simplicity is the ultimate sophistication."
- Leonardo da Vinci

Abstract

This thesis targets two open problems in the literature related to the area of data compression in memory hierarchies as a means of expanding the capacity without physically adding more memory.

For the first part, as computational demands continue to increase, hybrid memory systems that integrate High-Bandwidth Memory (HBM) as Near Memory (NM) and DRAM as Far Memory (FM) present a compelling solution for achieving high-bandwidth, large-capacity, and cost-effective main memory. However, existing flat hybrid memory architectures suffer from either excessive swap traffic or underutilized NM capacity. To address these challenges, this paper introduces HMComp, a novel flat hybrid-memory architecture that uses memory compression techniques to optimize NM usage, creating cache space for FM. By dynamically repurposing the freed-up NM capacity as a cache for FM data, HMComp effectively reduces swap traffic while maintaining full memory capacity. Additionally, a carefully designed metadata layout ensures that metadata is stored in the low-cost FM, preserving valuable NM capacity for critical application data. Experimental results demonstrate that HMComp achieves up to a 22% improvement in single-thread performance (13% on average) and reduces swap-related traffic by up to 60% (41% on average) compared to traditional flat hybrid memory systems.

The second part of this work presents COMPAT, an advanced memory compression framework designed to overcome the limitations of traditional approaches that use memory compression to expand memory capacity. Conventional memory compression frameworks often rely on a limited set of fixed compression unit sizes and struggle with misaligned block sizes relative to application data structures, resulting in suboptimal memory expansion. COMPAT introduces a novel object-based compression technique that enables fine-grained compression unit sizing without the overhead typically associated with dynamic size adjustments. The framework features a load-store unit design capable of inferring object structures with minimal changes to the memory system and no modifications to the instruction set architecture. COMPAT additionally offers fine-grain compression unit sizes and resolves the issue of compression unit size alteration overhead by a compression stabilization technique. This technique does not compress pages until the compression unit sizes of blocks in a page stabilize. Experimental results show that COMPAT achieves a 30% improvement in compression ratio and a 35% boost in performance over existing memory compression solutions. With the compression stabilization technique, COMPAT reduces the compression unit size alteration ratio from 22.5% (without stabilization) to 0.9%.

Keywords: Memory Compression, Hybrid Memory Management, Object-Level Compression

Acknowledgment

First and foremost, I would like to express my deepest and sincerest gratitude to my advisor, Professor Per Stenström, for his invaluable guidance and unwavering support throughout my studies. His research acumen, vision, and relentless motivation have been a constant source of inspiration, providing me with a methodological foundation to conduct research and present my work. I am truly lucky to have had Professor Per as my advisor.

I am also immensely grateful to my co-advisor, Angelos Arelakis, for his insightful feedback from both research and industry perspectives. My heartfelt thanks go to my examiner, Professor Pedro Trancoso, for his valuable advice and guidance, both in my research and beyond. I would also like to extend my gratitude to Professor Miquel Pericàs for his support in presenting my first paper. Additionally, I am thankful to my line manager, Arne Linde, as well as Clara Oders and Monica Månhammar, for their continuous support.

A special thank you goes to my friends at Chalmers—Konstantinos, Piyumal, Jing, Minyu, Magnus, Neethu, Panagiotis, Feng, Yixu, Mateo, Fareed, Hari, and many others—for their companionship, encouragement, and the memorable moments we shared throughout this journey. I am especially grateful to Kons for his insightful sharing on compilers and prefetching.

Lastly, I would like to express my deepest gratitude to my parents and my wife for their unwavering support and encouragement. I am also profoundly grateful to the Wallenberg AI, Autonomous Systems, and Software Program (WASP) for providing me with the opportunity to expand my horizons, gain invaluable knowledge, and forge lasting friendships with researchers.

This work has been supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP),funded by Knut and Alice Wallenberg Foundation, and by the Swedish Foundation for Strategic Research under contract number CHI19-0048. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

List of Publications

Appended publications

This thesis is based on the following publications. References to the papers are made using Roman numerals associated with the papers:

I. Qi Shao, Angelos Arelakis and Per Stenström
"HMComp: Extending Near-Memory Capacity using Compression in Hybrid Memory"
In Proceedings of the *38th ACM International Conference on Supercomputing (ICS)*, pages 74–84, June 2024.

II. Qi Shao, Per Stenström
"COMPAT: A Compression Framework with Fine-Grain and Object-Level Compression Units"
Submitted for publication.

Part of the contributions resulted in the following inventions.

[a] Qi Shao, Angelos Arelakis and Per Stenström
Hybrid Memory Compression
Patent number: 2450610-7
Filed on June 9, 2024 at the Swedish Patent Office

[b] Qi Shao and Per Stenstrom
COMPAT: A Compression Framework with Fine-Grain and Object-Level Compression Units
Patent number: 2550203-0
Filed on March 5, 2025, provisionally, at the Swedish Patent Office

Contents

Chapter 1

Introduction

1.1 Background

A modern computer system consists of one or more processing units and a hierarchical memory subsystem. This memory hierarchy typically includes an on-chip cache hierarchy of, say, three levels (L1, L2, and L3), off-chip memory, and disk storage. Here, the level closest to the processing units is referred to as level 1 (L1), and the one farthest away from the processing units is referred to as level 3 (L3). In some architectures, off-chip memory may be further divided into multiple levels to optimize performance, known as hybrid memory.

When the processing unit requires data, it first checks the L1 cache. If the data are found, the access latency is short, often just a few nanoseconds. If the data are not present, the request proceeds to the L2 cache, then the L3 cache, and subsequently to off-chip memory, if necessary. As the memory level moves further away from the processor, access time increases. In modern CPUs, L1 cache access times are typically on the order of a few nanoseconds, while L2 and L3 caches exhibit higher latencies measured in tens of nanoseconds. Accessing off-chip memory may take tens to hundreds of nanoseconds, and fetching data from disk can take milliseconds due to mechanical and I/O overhead.

One straightforward approach to mitigate limited memory capacity is to increase the physical size of on-chip caches and off-chip memory. However, increasing the amount of physical memory leads to increased power consumption and increased manufacturing costs. To address these challenges, state-of-the-art memory systems [1, 2] increasingly adopt compression techniques. By compressing data in on-chip caches or off-chip memory, one can effectively expand available memory capacity without the need for significant hardware modifications, improving performance and resource utilization in modern computing systems.

The literature on cache compression is rich (e.g., [3–8]). Cache compression techniques aim to store frequently accessed data more efficiently within the cache, thereby reducing the number of cache misses and improving performance. Similarly, memory compression (e.g., [9–15]) aims at expanding the effective capacity of off-chip memory by compressing memory data, thereby reducing the number of costly page faults that require disk access. Furthermore, modern computer systems often adopt hybrid memory architectures that combine high-

bandwidth memory (HBM) technology with dynamic random-access memory (DRAM) technology. Achieving an optimal balance between these two distinct memory types presents a significant challenge due to their differing performance characteristics and costs.

1.2 Problem Statement

Modern computer systems use hybrid memory architectures, with High Bandwidth Memory (HBM) as near memory (NM) and DRAM as far memory (FM). Current frameworks [16–19] swap pages between these two memory levels to balance performance and capacity, but suffer from traffic overhead. Frameworks [20, 21] allocate cache in near memory to reduce swapping overhead by caching bandwidth-demanding data from far memory. However, this wastes near-memory space due to static cache allocation. The problem I address is the following:

How can we use state-of-the-art compression techniques to free up cache space in near memory?

Limited memory capacity often forces requests to access slower memory levels, such as the disk, when off-chip memory is full. This leads to costly page-swapping overhead. Compression frameworks [10, 14, 15, 22] use compression to expand off-chip memory and reduce page faults, improving performance. However, these frameworks rely on fixed block sizes (e.g., 64 bytes), which do not align well with object-oriented workloads. Compressing at the object level while reducing duplication offers better efficiency. However, the existing object-based framework [15] requires complex changes to the memory architecture and modifications at runtime which are not compatible with conventional cache hierarchies.

How can we enable object-level compression and allow fine-grain compression unit sizes in a conventional cache hierarchy?

1.3 Thesis Contributions

This thesis is based on two papers.
Paper I addresses the first problem and the main contributions are:

- We propose HMComp, a novel hybrid memory architecture that exposes the combined capacity of NM and FM to the system and employs data compression techniques to dynamically free up NM capacity, enabling it to cache bandwidth-intensive blocks from FM.
- We introduce a novel metadata management scheme that minimizes overhead. Compressed NM blocks are co-located with cached-compressed blocks, simplifying address translation and reducing metadata complexity.

Paper II addresses the second problem and the main contributions are:

- Conventional wisdom limits the number of compression unit sizes due to three sources of overhead: address calculation, metadata and compression unit size alteration. We show that the first two sources of overhead are

not fundamental. COMPAT offers a solution to the third overhead and shows that substantially more compression sizes without overhead can translate into higher compression ratios and more memory expansion.

- COMPAT supports object-level compression by proposing a novel load-store unit design, without any changes to neither the ISA nor the memory hierarchy.

The rest of the thesis is organized as follows. In Chapter 2, a summary of each paper is presented. Finally, Chapter 3 concludes the thesis, and discusses some possible future research directions.

Chapter 2

Summary of the Papers

2.1 Summary of Paper I

The limited bandwidth of Dynamic Random-Access Memory (DRAM) has long been a bottleneck in modern computing systems, particularly for data-intensive applications. To address this limitation, heterogeneous memory systems featuring a two-level main-memory hierarchy have gained prominence. A notable example of such systems is a two-level memory architecture, where High-Bandwidth Memory (HBM) serves as the first level, referred to as Near Memory (NM), while DRAM functions as the second level, known as Far Memory (FM). HBM offers significantly higher bandwidth compared to DRAM; for instance, HBM3E can deliver up to 1229 GB/s [23], whereas DDR5 provides only around 20–40 GB/s [24]. While the enhanced bandwidth of HBM can greatly benefit data-intensive workloads, its cost remains substantially higher than that of DRAM. Consequently, the primary objective of such hybrid memory systems is to leverage the high bandwidth of HBM while maintaining a cost closer to that of DRAM.

Existing research has explored two primary approaches for managing hybrid memory systems: *cached* and *flat* hybrid memory architectures. In the cached approach, NM functions as a cache for FM, operating transparently to the operating system (e.g., [25–29]), wasting the capacity of NM. In contrast, flat hybrid memory architectures treat both NM and FM as part of a unified physical address space. In such systems, bandwidth-intensive pages are typically allocated to NM (e.g., HBM), while less bandwidth-sensitive pages are mapped to FM (e.g., DRAM) [16, 17, 30–33]. However, the dynamic adjustment of page mappings in flat hybrid memory systems introduces significant overhead, both in terms of operating system involvement and the data traffic generated during page swaps.

To address this overhead, prior work has proposed hardware-based remapping mechanisms with finer allocation granularity [18–21, 34–36]. However, along with the overhead of swapping traffic, tracking access patterns at finer granularities requires substantial metadata, which can reduce the effective capacity of NM and require significant on-chip memory resources. This trade-off between granularity and metadata overhead remains a critical challenge in hybrid memory system design.

In Paper I, we introduce HMComp, a Hybrid Memory Compression Framework, which addresses these challenges through a novel approach. Unlike prior methods, HMComp (1) presents the full combined capacity of NM and FM in a flat memory model and (2) dynamically creates a cache space within NM by leveraging the space freed through data compression. This dual approach enables efficient utilization of NM's bandwidth while minimizing cost and overhead.

The granularity at which bandwidth demand is monitored within a page plays a crucial role in determining the metadata overhead and system performance. Finer granularity increases metadata requirements, while coarser granularity can enhance performance through prefetching when swapping or caching larger data units. To balance this trade-off, HMComp tracks data at the *subpage* level (2 KB) and performs swapping/caching at the *superblock* level (512 B). This design reduces tracking overhead while capitalizing on the benefits of spatial locality and prefetching.

HMComp adopts the *congruence group* organization from the prior work CAMEO [19], where a page in NM is mapped to a set of FM pages. Similar to a direct-mapped cache, a set of FM pages is associated with a single page in NM. Bandwidth-demanding pages in FM are either swapped with pages in NM within the same *congruence group* or cached in the space created by compressing NM pages. This approach ensures efficient utilization of NM's bandwidth while maintaining a simple and scalable mapping mechanism.

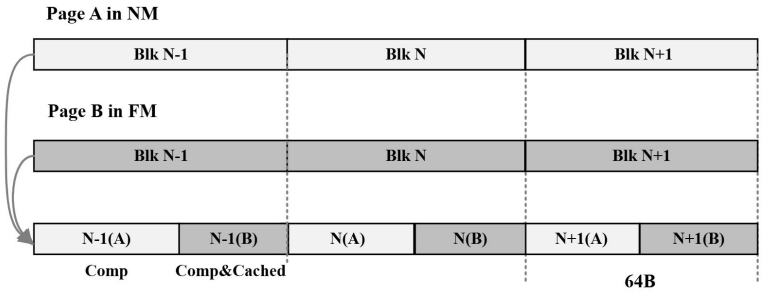

Figure 2.1: Combing compressed & cached blocks in congruence groups. Comp stands for compressed.

To identify bandwidth-intensive data in FM, HMComp monitors memory requests at the Last-Level Cache (LLC). When a subpage mapped to FM is accessed, a reference counter tracks the number of accesses. If the counter exceeds a predefined threshold, the subpage is classified as bandwidth-demanding and becomes a candidate for swapping or caching in NM. At this point, the subpage is transitioned to swap mode, and HMComp attempts to create cache space in NM through compression. This dynamic allocation mechanism ensures that bandwidth-critical data is efficiently migrated to NM, optimizing system performance while minimizing swapping traffic overhead.

To evaluate the performance of HMComp, we use the Gem5 [37] simulator with SimPoint [38] technology to select representative slices from the SPEC2017 [39] benchmark. We compare HMComp's performance against systems operating in flat and swap modes, as well as Hybrid2 and Baryon.

Experimental results show that for bandwidth-sensitive workloads such as `mcf`, `gcc`, `fotonik`, `wrf`, and `lbm`, HMComp improves single-thread performance by over 20%. Additionally, for workloads with a memory compression ratio above two, such as `mcf`, `gcc`, `fotonik`, and `wrf`, HMComp enhances performance by creating additional cache space in HBM through compression. Cache space created from compression in these benchmarks results in a reduction of swap traffic by up to 60% in `mcf`.

However, in `lbm`, which has a lower compression ratio of 1.33, the speedup is slightly lower than that of Hybrid2 and Baryon, as these schemes allocate cache statically. However, HMComp can allocate a separate cache in NM to achieve competitive performance in `lbm`.

In summary, HMComp achieves a single-thread performance speedup of up to 22%, averaging 13%, while reducing traffic caused by swapping by up to 60%, with an average reduction of 41%, compared to flat hybrid memory designs.

2.2 Summary of Paper II

The rapid growth in computational performance has brought a surge in the memory demands of modern applications. Expanding memory capacity without adding more DRAM presents a valuable opportunity to reduce the total cost of ownership (TCO) in data centers. Memory compression has emerged as a promising solution to enhance memory capacity and bandwidth, prompting extensive research into various memory compression frameworks.

Despite advancements, existing memory compression frameworks generally rely on a limited set of fixed compression unit sizes for compressed memory blocks. For example, Compresso [14], a state-of-the-art main memory compression framework, offers a choice of four compression unit sizes. However, application workloads typically exhibit significant variation in optimal compression unit sizes, resulting in suboptimal compression ratios when constrained by fixed-size limitations. Previous frameworks identified three primary challenges preventing the adoption of fine-grained compression unit sizes:

- **Address calculation overhead:** Frameworks such as LCP [10] and Compresso [14] compute block locations by adding the compression unit sizes of preceding blocks within the same page. These frameworks argue, without strong empirical evidence, that fine-grained compression unit sizes could lead to prohibitive address calculation overhead.

- **Metadata overhead:** Logging fine-grained compression unit sizes for each block in metadata could theoretically increase metadata overhead significantly. Studies such as [7] and [19] make this claim but lack concrete evidence to support the assertion that using eight or more compression unit sizes would incur prohibitive costs.

- **Compression-Unit-Size alteration overhead:** When a block is modified during a write-back operation, its compression unit size may change. This alteration necessitates shifting subsequent blocks to accommodate the new size, potentially introducing significant overhead.

In Paper II, we demonstrate that two of these limitations, address calculation and metadata overhead, are not fundamental barriers to adopting fine-grained compression unit sizes. However, the third limitation, compression-unit-size alteration overhead, is a challenge. To address this, we introduce the COMPAT framework, which employs a novel technique called *Compression Stabilization* to mitigate alteration overheads effectively.

Another limitation of prior memory compression frameworks, is their use of fixed-size memory blocks for compression. Emerging object-based workloads often contain data that is misaligned with these fixed block sizes. Compressing data at the object level, rather than at the block level, can eliminate redundancy between objects and achieve higher compression ratios.

Prior art, COCo [15] achieves object-level compression by reconfiguring the memory hierarchy into scratchpad memories and introducing dedicated instructions for object allocation. In Paper II, we offer a simple yet powerful load-store unit design that dynamically infers the layout of objects. This allows the compressor to operate at the object level, compatible with block-based cache hierarchies.

In Paper II, we analyze the three types of overhead individually through a combination of circuit-design analysis and back-of-the-envelope calculations. Our findings reveal that the latency of the address calculation in state-of-the-art adder designs can be as low as 0.3 ns, which is consistent with the conclusion presented in Compresso [14]. Additionally, supporting finer-grained compression unit sizes, such as an 8-byte stride, introduces only a 0.39% increase in metadata overhead. These results demonstrate that the first two overheads—address calculation latency and metadata overhead—are not fundamental barriers to implementing fine-grained compression.

However, our evaluation highlights that the third overhead, compression-unit-size alteration overhead, is a real issue. We find that the overhead rises significantly from 10% for frameworks with four compression unit sizes to 23% for frameworks with nine. This increase underscores the challenges of achieving fine-grained compression without introducing prohibitive overhead.

To address this challenge, COMPAT introduces a compression stabilization technique that leverages write-operation counters and write-alteration counters for each memory page. The write-operation counter increments with each write-back operation, and if the compressed size of a block changes, the write-operation count is transferred to the write-alteration counter. A threshold is established as the average of write-alteration counters of multiple pages. Based on the threshold, pages remain uncompressed until the write-operation count exceeds this threshold, at which point the page is compressed. This mechanism avoids compression during periods of frequent compression-unit-size alteration.

In object-oriented workloads, compressing data at the object level can achieve higher compression ratios by eliminating redundancy between objects. The primary challenge lies in accurately determining the size of the object. Unlike COCo, which modifies the runtime system and compiler to infer object sizes, COMPAT takes a more streamlined approach. It leverages the insight that instructions in object-oriented workloads typically traverse objects within a page during initialization or updates.

Drawing inspiration from PC-based stride prefetchers [40], COMPAT identifies object sizes by calculating the address differences between accesses to the same page. Once the object size is determined, COMPAT informs the compressor to enable object-level compression. By applying XOR operations to reduce redundancy between objects, similar to the technique used in BCD [22], COMPAT not only achieves a high object-level compression ratio but also maintains compression and decompression latency comparable to Compresso.

To evaluate the performance of COMPAT, we adopt the same methodology as in Paper I to collect memory compression ratios. Performance speedup is then measured by calculating the instructions per cycle (IPC) under a constrained memory capacity budget, adjusted based on the memory compression ratio on a real machine.

We compare COMPAT's performance and overhead against four baseline systems. Baseline 1 (BL1) represents a system without memory compression. LCP [10] is a state-of-the-art design that compresses blocks into a fixed compression unit size. Compresso [14] improves upon this by supporting four different compression unit sizes. Baseline 2 (BL2) is a system that compresses 64-byte blocks to sizes ranging from 0 to 64 bytes in 8-byte increments. However, BL2 lacks both compression stability mechanisms and object-based compression.

Experimental results show that, considering compression ratio improvements, COMPAT outperforms other compression frameworks, achieving a 44.2%, 27.9%, and 8.5% increase over LCP, Compresso, and BL2, respectively. For workloads dominated by objects, such as `deepsjeng`, `gcc`, `lbm`, `leela`, `mcf`, `parest`, and `xalancbmk`, COMPAT improves the memory compression ratio, with a geometric mean improvement of 13.0%, reaching up to 43.4% in `mcf` compared to BL2.

This enhanced compression ratio translates into an average speedup of 46.5%, 34.5%, and 15.5% over LCP, Compresso, and BL2, respectively. Furthermore, compared to BL2, COMPAT reduces the compression unit size alteration overhead from 22.5% to 0.9%, making this overhead negligible.

Chapter 3

Conclusions and Future Work

Paper I introduces HMComp, a hybrid memory compression framework that effectively addresses the challenges of limited DRAM bandwidth in data-intensive applications. By leveraging a two-tier memory architecture with High-Bandwidth Memory (HBM) as Near Memory (NM) and DRAM as Far Memory (FM), HMComp combines the benefits of flat memory architectures with dynamic caching enabled through data compression. The innovative approach of dynamically creating a cache within NM by compressing data not only increases bandwidth utilization but also minimizes cost and metadata overhead.

Paper II presents COMPAT; a framework to expand memory capacity in general memory system, which introduces two orthogonal techniques: fine-grained compression unit sizes and object-based compression. Through a detailed circuit design analysis, COMPAT debunks prior misconceptions about overheads for the size of the fine grain compression unit. However, Paper II identifies compression-unit-size alteration overhead as a genuine challenge and addresses it through the compression stabilization technique. Furthermore, COMPAT proposes a novel load-store unit design that dynamically infers object sizes, enabling object-level compression in conventional block-based cache hierarchy without the need to add custom instructions. This dual approach improves the compression ratio.

For future work, firstly, during my prior research, I identified a gap in the availability of open-source simulators for managing hybrid memory systems, particularly in caching and swap modes. When building a hybrid memory system with Gem5, I encountered several modeling challenges. For instance, naively blocking pages for caching or swapping can sometimes degrade performance rather than improve it.

Although extensive research has been conducted on managing bandwidth-intensive data for caching and swapping, the need for a dedicated open-source simulator remains an open issue. By analyzing simulation results, we can pinpoint performance bottlenecks. With these insights, I hope to develop optimizations and contribute an open-source simulator that can serve as a valuable tool for the research community.

Secondly, another promising research avenue lies in optimizing GPU memory. As the parameters of large language models (LLMs) continue to grow exponentially, the demand for GPU memory increases accordingly. Implementing memory optimization techniques, such as compression, during training and inference could reduce the GPU memory footprint and alleviate GPU traffic. These advancements would help address the challenges of memory-bound GPU applications, leading to tangible improvements in both performance and efficiency.

Bibliography

[1] NVIDIA Corporation, "nvCOMP: GPU-Accelerated Lossless and Hybrid Compression Library," 2025, accessed: March 5, 2025. [Online]. Available: https://developer.nvidia.com/nvcomp

[2] A. Buyuktosunoglu, D. Trilla, B. Abali, D. Berger, C. Walters, and J.-S. Lee, "Enterprise-class cache compression design," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 996–1011.

[3] S. Sardashti and D. A. Wood, "Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: Association for Computing Machinery, 2013, p. 62–73. [Online]. Available: https://doi.org/10.1145/2540708.2540715

[4] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, p. 150–159, Nov. 2000. [Online]. Available: https://doi.org/10.1145/378995.379235

[5] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 331–342.

[6] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: a hybrid cache compression method for selection of data-type-specific compression methods," ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 38–49. [Online]. Available: https://doi.org/10.1145/2830772.2830823

[7] S. Hong, B. Abali, A. Buyuktosunoglu, M. B. Healy, and P. J. Nair, "Touché: Towards ideal and efficient cache compression by mitigating tag area overheads," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 2019, pp. 453–465. [Online]. Available: https://doi.org/10.1145/3352460.3358281

[8] 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, no. 3, Sep. 2016. [Online]. Available: https://doi.org/10.1145/2976740

[9] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 74–85.

[10] G. Pekhimenko, T. C. Mowry, and O. Mutlu, "Linearly compressed pages: A main memory compression framework with low complexity and low latency," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 489–489.

[11] J. Dusser and A. Seznec, "Decoupled zero-compressed memory," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 77–86. [Online]. Available: https://doi.org/10.1145/1944862.1944876

[12] 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)*, 2016, pp. 329–340.

[13] C. Qian, L. Huang, Q. Yu, Z. Wang, and B. Childers, "Cmh: compression management for improving capacity in the hybrid memory cube," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, ser. CF '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 121–128. [Online]. Available: https://doi.org/10.1145/3203217.3203235

[14] E. Choukse, M. Erez, and A. R. Alameldeen, "Compresso: Pragmatic main memory compression," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 546–558.

[15] P.-A. Tsai and D. Sanchez, "Compress objects, not cache lines: An object-based compressed memory hierarchy," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 229–242.

[16] C. Chou, A. Jaleel, and M. Qureshi, "Batman: Techniques for maximizing system bandwidth of memory systems with stacked-dram," in *MEMSYS*. ACM, 2017, pp. 268–280.

[17] P. Duraisamy, W. Xu, S. Hare, R. Rajwar, D. Culler, Z. Xu, J. Fan, C. Kennelly, B. McCloskey, D. Mijailovic, B. Morris, C. Mukherjee, J. Ren, G. Thelen, P. Turner, C. Villavieja, P. Ranganathan, and A. Vahdat, "Towards an adaptable systems architecture for memory tiering at warehouse-scale," in *ASPLOS 2023*. New York, NY, USA: ACM, 2023, p. 727–741.

[18] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 13–24.

[19] C. C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 1–12.

[20] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Hybrid2: Combining caching and migration in hybrid memory systems," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 649–662.

[21] Y. Li and M. Gao, "Baryon: Efficient hybrid memory management with compression and sub-blocking," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 137–151.

[22] S. Park, I. Kang, Y. Moon, J. H. Ahn, and G. E. Suh, "Bcd deduplication: effective memory compression using partial cache-line deduplication," ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 52–64.

[23] Micron Technology, Inc., "HBM3E Product Brief," 2024. [Online]. Available: https://www.micron.com/content/dam/micron/global/public/documents/products/product-flyer/hbm3e-product-brief.pdf

[24] ——, "DDR5: More Than a Generational Update," 2024. [Online]. Available: https://www.micron.com/content/dam/micron/global/public/products/white-paper/ddr5-more-than-a-generational-update-wp.pdf

[25] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 25–37.

[26] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 211–222.

[27] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 1–14.

[28] Y. Kim, H. Kim, and W. J. Song, "Nomad: Enabling non-blocking os-managed dram cache via tag-data decoupling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 193–205.

[29] F. Hameed and J. Castrillon, "Rethinking on-chip dram cache for simultaneous performance and energy optimization," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 362–367.

[30] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in *2015 IEEE 21st*

International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 126–136.

[31] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "Dynamically adapting page migration policies based on applications' memory access behaviors," in *ACM Journal on Emerging Technologies in Computing Systems*, 2021, pp. 1–24.

[32] ——, "Chameleon: A dynamically reconfigurable heterogeneous memory system," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 533–545.

[33] M. Islam, S. Adavally, M. Scrbak, and K. Kavi, "On-the-fly page migration and address reconciliation for heterogeneous memory systems," vol. 16, no. 1, pp. 1–27, 2020.

[34] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "Silc-fm: Subblocked interleaved cache-like flat memory organization," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 349–360.

[35] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 433–444.

[36] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Llc-guided data migration in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 932–942.

[37] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[38] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program phase analysis," *Journal of Instruction Level Parallelism*, vol. 7, no. 4, pp. 1–28, 2005.

[39] S. P. E. Corporation, "Spec cpu2017." *Journal of Instruction Level Parallelism*, 2017.

[40] T.-F. Chen and J.-L. Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 609–623, 1995.