CHALMERS
UNIVERSITY OF TECHNOLOGY

HMComp: Extending Near-Memory Capacity using Compression in Hybrid Memory

Downloaded from: https://research.chalmers.se, 2024-07-17 18:14 UTC

Citation for the original published paper (version of record):

Shao, Q., Arelakis, A., Stenström, P. (2024). HMComp: Extending Near-Memory Capacity using Compression in Hybrid Memory. Proceedings of the International Conference on Supercomputing: 74-84. http://dx.doi.org/10.1145/3650200.3656612

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

HMComp: Extending Near-Memory Capacity using Compression in Hybrid Memory

Qi Shao
qisha@chalmers.se
Chalmers University of Technology
Gothenburg, Sweden

Angelos Arelakis
angelos.arelakis@zptcorp.com
ZeroPoint Technologies
Gothenburg, Sweden

Per Stenström
pers@chalmers.se
Chalmers University of Technology
ZeroPoint Technologies
Gothenburg, Sweden

ABSTRACT

Hybrid memories, especially combining a first-tier *near* memory using High-Bandwidth Memory (HBM) and a second-tier *far* memory using DRAM, can realize a large and low cost, high-bandwidth main memory.

State-of-the-art hybrid memories typically use a flat hierarchy where blocks are swapped between near and far memory based on bandwidth demands. However, this may cause significant overheads for metadata storage and traffic. While using a fixed-size, near-memory cache and compressing data in near memory can help, precious near-memory capacity is still wasted by the cache and the metadata needed to manage a compressed hybrid memory.

This paper proposes HMComp, a flat hybrid-memory architecture, in which compression techniques free up near-memory capacity to be used as a cache for far memory data to cut down swap traffic without sacrificing any memory capacity. Moreover, through a carefully crafted metadata layout, we show that metadata can be stored in less costly far memory, thus avoiding to waste any near-memory capacity. Overall, HMComp offers a speedup of single-thread performance of up to 22%, on average 13%, and traffic reduction due to swapping of up to 60% and by 41% on average compared to flat hybrid memory designs.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**.

KEYWORDS

Hybrid Memory, Memory Management, Memory Compression, HBM

ACM Reference Format:
Qi Shao, Angelos Arelakis, and Per Stenström. 2024. HMComp: Extending Near-Memory Capacity using Compression in Hybrid Memory. In *Proceedings of the 38th ACM International Conference on Supercomputing (ICS '24), June 04–07, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3650200.3656612

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICS '24, June 04–07, 2024, Kyoto, Japan
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0610-3/24/06
https://doi.org/10.1145/3650200.3656612

1 INTRODUCTION

Dynamic random-access memory (DRAM) is plagued by limited bandwidth. To mitigate it, heterogeneous memory systems consisting of a two-level main-memory hierarchy, a.k.a. *hybrid memory*, is an attractive way of addressing this deficiency. In this paper, we consider two-tier, hybrid memories with High-Bandwidth Memory (HBM) being the first-tier or *Near Memory* (NM) and DRAM being the second-tier or *Far Memory* (FM). HBM typically offers sixteen times higher bandwidth than DRAM [25, 29] to accommodate the bandwidth needed by data-intensive applications but the cost of HBM is substantially higher than DRAM. Consequently, this type of hybrid memory can provide a main memory that matches the high bandwidth of HBM, possesses a size equivalent to DRAM, and maintains a cost that is almost as low as that of DRAM.

Prior art has investigated two broad approaches to manage hybrid memories: *cached* and *flat* hybrid memories. In a cached hybrid memory, NM is used as a cache for FM, managed transparently to the operating system [9, 13, 14, 16, 19, 21, 28, 35]. However, for a hybrid memory using HBM and DRAM as NM and FM, respectively, as we do in this paper, the amount of DRAM is often a *small* factor, say eight, more than the amount of HBM. Hence, by not exposing NM to the (operating) system, a significant amount of memory capacity is wasted.

In a flat hybrid memory, NM as well as FM contribute to the flat physical address space making the entire hybrid-memory capacity available to the (operating) system. Here, bandwidth-demanding pages mapped in FM (e.g., DRAM) are typically swapped by less bandwidth-demanding pages residing in NM (e.g., HBM) [6, 8, 11, 17, 18, 22]. Unfortunately, changing the page mapping entails significant operating-system induced overhead along with the traffic overhead of swapping pages. To reduce the former type of overhead, prior art has proposed remapping mechanisms at the hardware level and have considered smaller grain sizes [7, 20, 26, 27, 30–33]. However, the metadata needed to track finer-grain access units for remapping can consume a significant portion of the NM capacity and can require significant on-chip memory resources for keeping remapping metadata.

Hybrid^2 [33] and Baryon [20] propose a middle ground between cache and flat hybrid memories by *statically* setting aside a portion of NM to cache data from FM to avoid costly swap operations. The rest of the NM capacity is available to the system in flat mode. While Hybrid^2 allows for fine-grain swapping and caching, metadata still consumes precious space in NM. Unlike Hybrid^2, Baryon additionally compresses data in NM to agnostically expand its capacity for cache or flat space. However, cache space is still *statically* set aside from the flat space and compression necessitates a staging area to

stabilize compressed data. ***Both approaches contribute to less NM capacity being available to the system.***

This paper proposes Hybrid Memory Compression (HMComp). Unlike previous work, HMComp (1) exposes the *entire* NM plus FM capacity in flat mode to the system and (2) dynamically exposes a cache in NM from capacity made available from compressing data in NM. The freed-up cache is used to bring more bandwidth-demanding FM data into NM to avoid costly swap operations. Through its novel management along with a carefully crafted metadata layout, metadata can be kept in FM. HMComp imposes virtually no area overhead in NM for metadata or for staging.

HMComp unlocks space for caching by selectively compressing data in NM. This is done by dynamically monitoring compressibility and bandwidth demands of fine-grain access units in FM. By allowing fine-grain management of hybrid memory, HMComp carefully manages compressed blocks in HBM with a minimum of metadata needed. For example, it compresses HBM blocks where they originally are mapped and uses surplus ECC bits (in HBM) in a clever way to locate a compressed block and eliminates the use of remap tables in NM altogether.

Contributions:

- HMComp – a novel hybrid memory architecture – that exposes the entire NM and FM capacity to the system. Further, HMComp compresses data to free up NM capacity (HBM) to cache bandwidth-demanding blocks from FM (DRAM). This includes techniques for dynamically assessing compressibility and bandwidth demand of fine-grain access units in FM.
- A novel metadata layout that keeps the overhead low by, among other techniques, placing compressed blocks at the same place as uncompressed blocks and using surplus ECC bits to store metadata in NM compactly. This eliminates the need for any metadata in NM and leads to modestly-sized on-chip memory structures to cache remapping metadata from FM.
- HMComp is quantitatively compared to state-of-the-art flat hybrid memory schemes. The evaluation shows that HMComp offers a speedup of single-thread performance of up to 22% and on average 13% and a swap traffic reduction of up to 60% and by 41%, on average, compared to flat hybrid memory designs. Finally, we show that a quite modest metadata cache of 256 KB suffices.

Outline of paper: We provide background and further motivation of the study in Section 2. In Section 3, we introduce HMComp. We move on to the experimental results presenting the methodology in Section 4 and the results in Section 5. Finally, we conclude in Section 6.

2 BACKGROUND

This section first establishes a baseline system for HMComp in Section 2.1. Then, we provide further motivation for the approach taken in HMComp in Section 2.2.

2.1 Baseline

The assumed baseline system is shown in Figure 1 with the additional structures needed for HMComp (Section 3) shaded in gray.

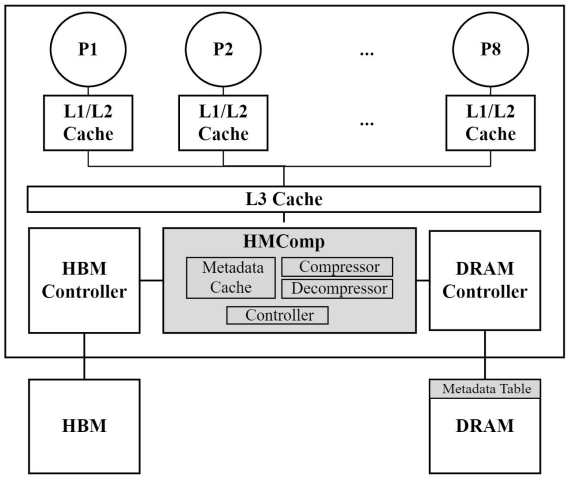


Figure 1: Baseline system with and without HMComp. HMComp extensions are marked in gray.

We consider a conventional multicore chip with a number of processors or cores (marked P), each connected to a private L1/L2 cache hierarchy and all cores and L1/L2 hierarchies are connected to a level-3 (L3) shared, last-level cache (LLC). LLC requests are routed to an HBM or DRAM controller, depending on whether a page is mapped to NM or FM (HBM and DRAM in this paper, respectively) as dictated by the virtual/physical page address mapping.

In our first baseline system, denoted *BL1*, the OS manages the HBM/DRAM in flat mode and interleaves pages using *congruence groups* [7] as shown in Figure 2. Here, the page size is *N* and page A is mapped to NM whereas pages B, C, D and E are mapped to FM, assuming a congruence group with one NM page and four FM pages. The structure of a congruence group can be likened to that of a direct-mapped cache, since pages B, C, D, and E are all vying for space within page A.

Our second baseline system, denoted *BL2* is built on top of *BL1*. When a page (or portion of it) mapped to FM is deemed bandwidth demanding, it is remapped to NM transparently to the operating system. This involves a *swap* operation with the page (or portion of it) congruent to it and located in NM. For example, block *K* in page C in FM is *congruent* with block *K* in page A in FM and the two blocks will be swapped with each other (see Figure 2).

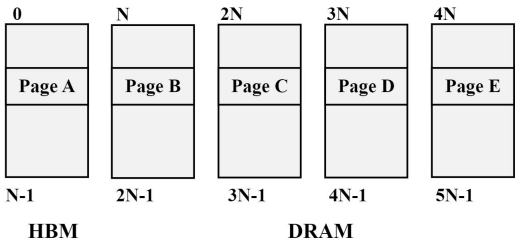


Figure 2: Congruence grouping of pages in HBM and DRAM.

The granularity chosen as the portion of a page to monitor bandwidth demand will determine the amount of metadata needed

to keep track of it. The finer the grain size the more metadata is needed. This will push towards larger grain sizes. On the other hand, a too large grain-size can lead to overfetching of data due to limited spatial locality, resulting in too high traffic overhead for swapping. Therefore, the trade-off when selecting a grain size is between metadata overhead and spatial locality, and grain sizes other than pages and blocks are considered in this paper. Throughout the paper, and to clarify terminology, we will deal with the following grain sizes of access units: *Pages*, *subpages*, *superblocks* and *blocks*, where the size of pages > subpages ≥ superblocks > blocks. As exemplary sizes of these access units, we will assume 4KB, 2KB, 512B and 64B for pages, subpages, superblocks, and blocks, respectively, if not stated otherwise.

2.2 Motivation

The baseline systems described in Section 2.1 are operated in flat mode meaning that BL1 as well as BL2 expose the entire NM as well as FM capacity to the operating system. Typically, when an access unit in FM, being a page, subpage or superblock, is deemed bandwidth demanding, it will be swapped by the corresponding congruent access unit in NM. The swapping of access units between NM and FM can cause overhead in terms of increased traffic. In a hybrid system with HBM (being NM) and DRAM (being FM), this can take away the bandwidth advantage and performance potential of such a hybrid memory.

To reduce the traffic overhead caused by swapping in flat hybrid memories, Hybrid^2 [33] proposes to *statically* set aside a portion of the NM capacity as a cache. This allows bandwidth demanding access units at the granularity of super-blocks (e.g., 256-B units) in FM to be cached in NM. Hybrid^2 saves traffic because caching an access unit, as opposed to swapping, leads to less traffic between NM and FM. However, statically setting aside NM capacity for caching nonetheless reduces the amount of NM capacity exposed to the system.

Like Hybrid^2 [33], Baryon [20] also sets aside a portion of the NM capacity as a cache *statically* to transform expensive swap to fast NM cache operations. But unlike Hybrid^2, Baryon uses compression techniques to expand the capacity of the flat as well as the cache area of NM. As Baryon is agnostic to the choice of compression algorithm, the cache portion of NM can be potentially expanded by the compression factor offered by the compression algorithm at hand. While Baryon potentially can also expand the flat portion of NM, this would require interventions with the operating system, such as with ballooning [34]. Baryon does not address this.

Hybrid^2 as well as Baryon carefully organize the metadata to locate whether fine-grain access units, in effect superblocks, are cached in NM or are in the flat space of NM or FM. However, metadata is stored in NM and consumes precious NM capacity. In addition, since compression is subject to changes in the size of a compressed access unit, Baryon lets newly compressed superblocks (called sub-blocks in Baryon terminology) stay in a staging area to stabilize. When the compression ratio has stabilized, superblocks are committed to the cache area in NM or in the flat area of NM or FM.

The bottom-line is that statically allocated NM caches in prior art reduce NM capacity. Moreover, metadata needed for remapping in Hybrid^2 as well as Baryon further reduces the available NM capacity. Finally, Baryon's approach to enable compression leads to further reduction of NM capacity through a staging area located in NM. This paper shows how HMComp can expose the *entire* NM and FM capacity to the system while offering a NM cache through freed up NM capacity using compression to improve performance of a hybrid memory.

3 HMCOMP: EXTENDING NEAR-MEMORY (NM) CAPACITY USING COMPRESSION

In this section, we present the detailed design of HMComp. Section 3.1 provides an overview of HMComp. Then, in Section 3.2, we describe the metadata layout followed by a detailed description of the operation of HMComp in Section 3.3.

3.1 HMComp Overview

The objective of HMComp is to free up capacity in NM using fast compression techniques and use the freed-up capacity to cache bandwidth-demanding FM blocks. Without loss of generality, NM in this paper uses HBM devices whereas FM uses DRAM devices.

Just like in the baseline systems in Section 2.1, pages are mapped by the operating system to NM and FM using congruence groups. The baseline is extended with a functional block, denoted HMComp which is gray-shaded in Figure 1. HMComp intercepts all LLC requests. Initially, HMComp adopts the policy of BL1 to all FM pages meaning that none of them are subject to swapping from the very start. This is referred to as *non-swap mode*. However, when a FM-mapped page is accessed, it will be tracked by a reference counter at the granularity of a subpage. The reference counter will be incremented for each access to said subpage. For as long as the reference counter is below a preset threshold (32 is chosen in the experimental results), all superblocks of the tracked subpage will be accessed from FM and will not be swapped. However, when the reference counter exceeds a preset threshold, we say that the subpage is *bandwidth demanding* and the subpage will turn into *swap mode*. From this point, all accessed FM superblocks associated with a subpage in swap mode will be swapped with their corresponding NM superblocks belonging to the same congruence group. For details, see Section 3.3.

Once a subpage switches to swap mode, attempts will be made to gain cache space in NM through compression. This is done by attempting to compress a FM superblock in swap mode together with its corresponding congruent NM superblock. We note that HMComp is agnostic to the choice of compression algorithm and that any fast compression algorithm in prior art can be used (e.g., [1–3, 5, 15, 24]).

When a FM superblock is requested, HMComp will also request the corresponding congruent NM superblock. Next, for each pair of blocks in the two superblocks, it will be decided whether these two blocks compress sufficiently well meaning that they can be accommodated within the same 64-B block frame. If a certain fraction of the blocks within a requested superblock is sufficiently well compressed, using above definition, the corresponding subpage is deemed to be in *cache/compress mode*. We have experimentally established that a fraction of seven blocks out of eight is a good

trade-off. All sufficiently well compressible blocks inside said superblock will be compressed. An uncompressible FM-mapped block will stay uncompressed in FM.

From now on, an attempt is made to compress all superblocks inside the subpage to be placed in NM. HMComp will then update the metadata table (for details, see Section 3.2) so that subsequent requests for the remapped superblocks are destined to NM with no involvement of FM. Otherwise, the subpage will remain in swap mode and the requested superblock will be swapped with the corresponding congruent superblock in NM (for details, see Section 3.3).

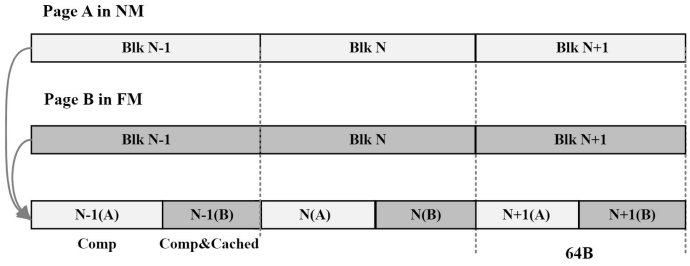


Figure 3: Combining compressed & cached blocks in congruence groups. Comp stands for compressed.

To see how compressible blocks are compressed, Figure 3 shows three contiguous compressed blocks (blocks $N-1, N$, and $N+1$) from two congruent pages A and B. Here, blocks $N-1, N$, and $N+1$ from page B in FM are compressed and stored together with their corresponding congruent blocks $N-1, N$, and $N+1$ from page A in NM. If the LLC subsequently requests block N in page B, HMComp will reroute the request to NM based on the metadata to the cached N^{th} block in page A. In the case that the FM block and the corresponding congruent NM block cannot fit into the 64-B block frame, the FM block will remain in FM and HMComp will verify the response from NM and then forward the request to FM (for details, see Section 3.3).

3.2 HMComp: Metadata Layout

For HMComp to decide which action to take for each LLC request, it uses a metadata cache. LLC requests will be routed either to NM or FM. In Section 3.2.1, we describe the layout of the metadata table. Section 3.2.2 describes the organization of the metadata cache and, finally, Section 3.2.3 describes the metadata needed in NM.

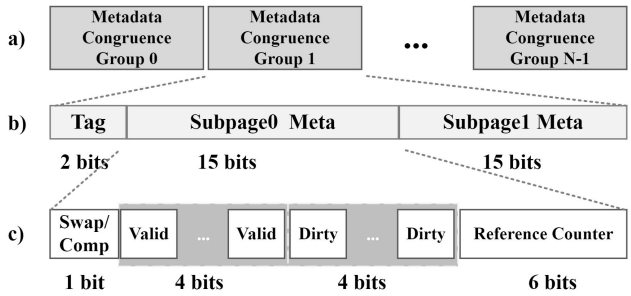


Figure 4: Metadata table layout with a) one entry per congruence group b) each entry has metadata for the two subpages of a referenced FM page and c) metadata for each subpage.

3.2.1 Metadata Table Layout. Recall that HMComp initially operates the hybrid memory in flat mode, where a page is mapped to NM or FM by the operating system. However, when the reference count of a referenced FM-mapped subpage exceeds a preset threshold, it will be remapped from FM to NM at the granularity of superblocks. From this point, requested FM-mapped superblocks belonging to that subpage will be swapped with the corresponding congruent NM-mapped superblock.

The layout of the metadata table is shown in Figure 4. The metadata table associates an entry with each congruence group, as shown in Figure 4a). Hence, it has as many entries as the number of pages in NM. It is stored in FM but cached in the metadata cache in HMComp (see Figure 1). The metadata entry for a congruence group is constructed to track one out of all FM pages belonging to a congruence group. Hence, as shown in Figure 4b) and assuming two subpages per page, a metadata entry for a congruence group needs a Tag of 2 bits to designate one out of four tracked FM pages in the congruence group and the two subpages (2 KB each) belonging to the tracked page (4 KB), with 15 bits of meta data for each subpage.

The 15-bit metadata field for each subpage is shown in Figure 4c). To the left, a single bit (Swap/Comp) together with the content of the reference counter (Reference Counter), designates whether the subpage is in *non-swap, swap* or *cache/compress* mode. If the reference count is below a preset threshold, the subpage is in non-swap mode and requests will be destined to NM or FM based on the virtual-to-physical address mapping. If the reference count is above a preset threshold, the Swap/Comp flag designates if the subpage is in swap mode (flag is set) or in cache/compress mode (the flag is reset). Next, there is one valid bit for each of the four superblocks (default size is 512 B) that belong to a subpage (default size is 2 KB). A valid superblock bit designates that the FM-mapped superblock is swapped (Swap/Comp bit set) or compressed in NM together with its corresponding NM-mapped superblock (Swap/Comp bit cleared). There are also 4 dirty superblock bits. Whenever a request is written back in cache/compress mode, the superblock dirty bit will be set. Finally, to the right in Figure 4c), the reference counter for a superblock uses 6 bits.

3.2.2 Metadata Cache. Recall that we assume that the metadata table is stored in FM. HMComp is configured to cache contents of the metadata table using a *metadata cache* as shown in Figure 1. As we will see in Section 5, a 256-KB metadata cache with 8-way associativity will impose negligible impact on performance. Each metadata cache entry (64 B) in the metadata cache contains 16 consecutive metadata entries (32 bits each). Thus, the metadata cache is indexed by an address corresponding to the requested congruence group, stripping out the least significant 4 bits. Given that the NM size is $C = 2^c$ and the page size is $P = 2^p$ there are $N = 2^{c-p}$ congruence groups. The congruence group can be stripped out from the physical page number taking the most significant c bits.

On a metadata cache hit, HMComp will update the reference counter and retrieve the request's corresponding metadata entry from the metadata cache. Conversely, on a metadata cache miss, HMComp will first evict an entry to make room for the requested metadata entry and, if needed, write back the evicted metadata

entry to FM. Then, the missing metadata entry from FM is fetched into the metadata cache.

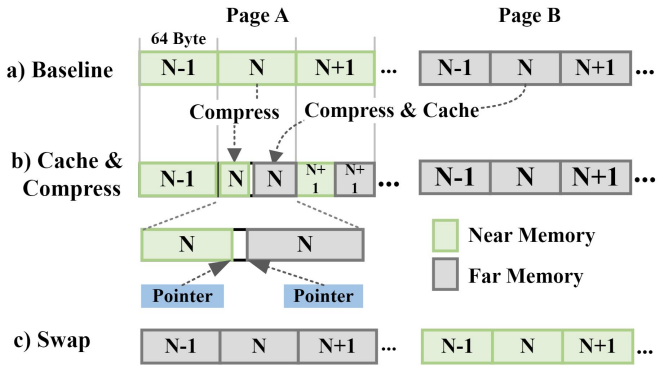


Figure 5: Block-level metadata in unused ECC bits. A green rectangle represents a NM block while a gray rectangle represents a FM block.

3.2.3 Near-Memory Metadata Support. Figure 5 shows the mapping of blocks inside two pages (A and B) in different operating modes. Here, pages A and B belong to the same congruence group and page A is mapped to NM whereas page B is mapped to FM. Recall that LLC block requests, as intercepted by HMComp, will be destined to NM in three cases with reference to Figure 5. The first case is in non-swap mode, when the block is mapped to NM. This corresponds to the baseline (BL1) in Figure 5a). The second case is in swap mode, when the Swap/Comp bit is set and the superblock valid bit is set. This corresponds to Swap in Figure 5c). Here, all blocks in page A have been swapped with the blocks in page B. Finally, the third case is in cache/compress mode when the Swap/Comp bit is reset and the superblock valid bit is set (see Figure 5). This corresponds to Cache & Compress in Figure 5b).

In cache/compress mode it is not certain that the requested block is in NM as a block may not compress sufficiently. Therefore, block-level information must be maintained in NM whether the FM-mapped block is compressed together with the corresponding congruent NM-mapped block. If not, the FM-block is placed in FM and the request has to be rerouted to FM.

HBM devices associate 16 ECC bits with each 32-B access unit [12]. When the NM-mapped and FM-mapped blocks in the same congruence group are compressed to fit into a 64-B block frame (two 32-B access units), we propose to use 6 unused ECC bits (out of 16) to encode the validity and size of the compressed NM-mapped and FM-mapped blocks. If the FM-mapped block is not compressed, it is stored in FM. This case is recorded by setting all six ECC bits to zero. If the FM and NM-mapped blocks are compressed, their compressed sizes will be recorded in the unused respective six ECC bits. For example, if the compressed size is 63 bytes, the 6 ECC bits will be encoded '111111' and if the compressed size is 2 bytes, the 6 ECC bits will be encoded '000010'. As shown in Figure 5, the NM block is placed starting at the original address of the block frame whereas the corresponding congruent FM-mapped block is mapped to the end of the block frame. 'Pointer' refers to the 6 ECC bits and are in effect interpreted as the location of the last byte. In the case ECC bits cannot be used, an alternative is to store metadata needed for compression, i.e., the size of the compressed block as part of the unused portions of the block. The only metadata needed outside of the HBM would be to designate whether or not the block is compressed, a single bit per block.

3.3 HMComp: Detailed Operation

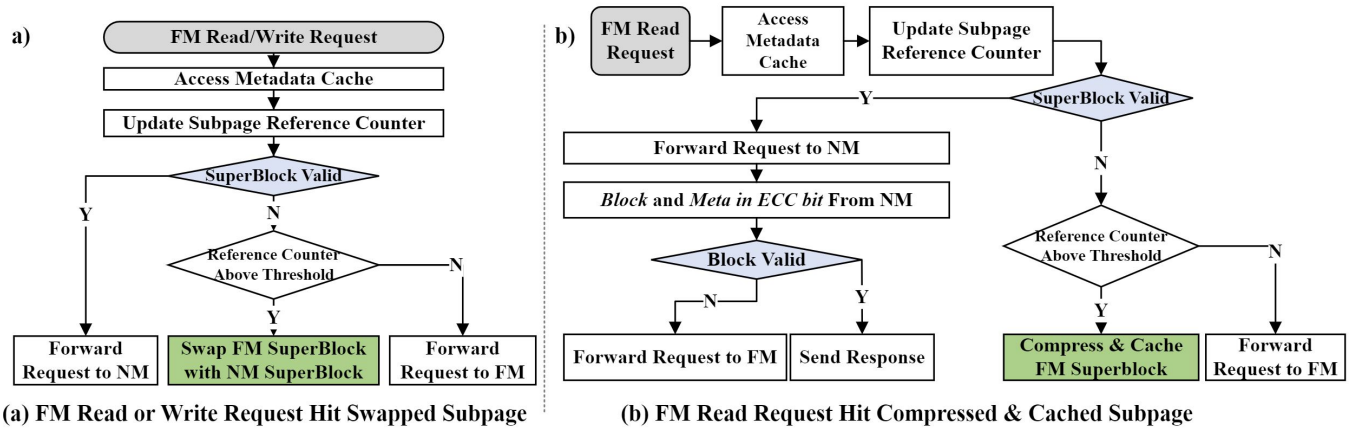
We now review in detail the operation of HMComp. Recall that HMComp initially, when in non-swap mode with respect to an LLC request, will send the LLC request to NM or FM depending on its virtual-to-physical mapping.

3.3.1 Mode Changes. A subpage will make a mode change from the non-swap mode to swap mode when the reference count exceeds a preset threshold. Figure 7 shows the process of going from swap mode to cache/compress mode for a subpage. For each subpage in swap mode and at each FM superblock request for that subpage, the first action is to request the FM superblock along with the corresponding congruent NM superblock. All blocks in the two superblocks will be pair-wise compressed. A pair of blocks that are compressed and can fit into a 64-B blockframe will be successfully compressed. If at least seven out of all eight blocks in a superblock are successfully compressed, the valid bit for said superblock will be set and the subpage will be in cache/compress mode. Otherwise, the subpage is set to swap mode.

3.3.2 Transaction Flow for Last-level Cache Requests. We now consider the transaction flow associated with LLC read or write requests to FM-mapped pages as shown in Figure 6 in the case the subpage is in swap mode (Figure 6a)) and cache/compress mode (Figure 6b)). As shown in Figure 6a), in swap mode, upon receiving a FM read or write request, HMComp will check whether the superblock is already in NM as a result of an earlier swap operation. If the superblock Valid bit is set, HMComp will forward the request to NM. If the valid bit is cleared, HMComp will swap the superblock in FM with the superblock in NM. This applies to both *read* and *write* FM requests in *swap* mode.

When considering the transactions of read requests to subpages in cache/compress mode, HMComp also first verifies the validity of the superblock. However, it is possible that some of the FM blocks within the superblock are cached in the compressed NM, while a few FM blocks still remain in FM. The latter applies to FM blocks that cannot be compressed and stored within a 64-B block frame along with the corresponding congruent and compressed NM block. For this reason, HMComp will examine the response from NM, including the data and relevant ECC bits. If the ECC bits are nonezero, the FM block is compressed and cached and HMComp will respond to LLC. If the ECC bits are zero, however, HMComp will forward the request to FM, as shown in Figure 6b).

The process for handling FM write request hits to subpages in cache/compress mode is illustrated in Figure 8. The issue here is that a written back block may, after compression, change in size and may not fit anymore. First, a test is carried out whether the size of the compressed written back block is greater than the already existing block. If not, the block is written back and the ECC bits are updated to reflect its new size. Meanwhile, the superblock dirty bit is set. However, if it exceeds the size but can still fit into the



(a) FM Read or Write Request Hit Swapped Subpage

(b) FM Read Request Hit Compressed & Cached Subpage

Figure 6: Transaction flow for a) LLC read/write requests in swap mode and b) read requests in cache/compress mode.

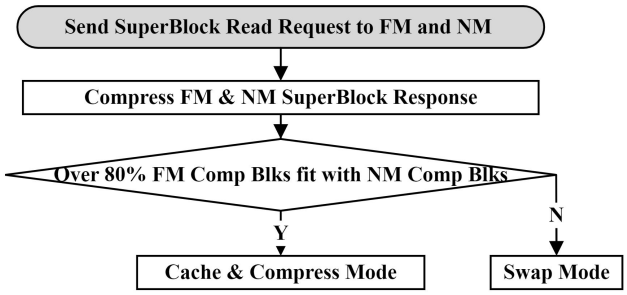


Figure 7: LLC read and write transactions for subpages in swap mode (left) and cache/compress mode (right).

64-B block frame together with the NM-mapped block, the block is written into NM and the process terminates. Finally, if it does not fit, the block will be forwarded to FM and the unused ECC bits of the congruent block in NM will be reset to reflect that it is not valid.

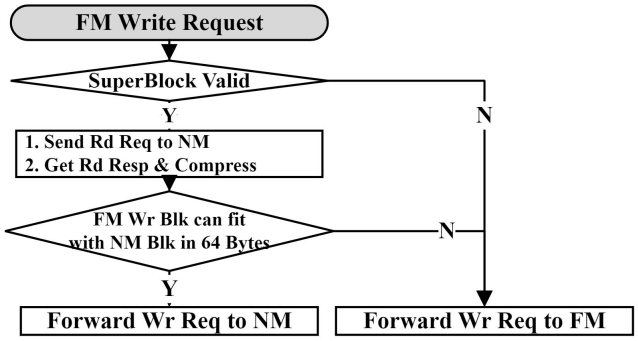


Figure 8: Transaction flow for write requests in cache/compress mode.

3.3.3 Other Operations Needed. When a page in a congruence group is being tracked, it can happen that another page in that same congruence group will be accessed. For as long as the the first page is not in swap mode, accesses to other pages inside the same congruence group will be disregarded. However, when the preset threshold is exceeded and the page will turn into swap mode, accesses by other pages inside the congruence group will decrement the reference counter. If it hits zero, the page will not be considered bandwidth demanding anymore and will make a transition from either cache/compress or swap mode to non-swap mode. This transition necessitates that all superblocks that have been potentially migrated to NM in non-compressed or compressed fashion must move back non-compressed to FM. We will call this operation *page consolidation*.

Page consolidation is also needed when the mapping is changed for a page by the operating system. Then, typically, TLB entries to the page have to be invalidated (TLB shootdown) and all blocks from said page must be evicted too. Page consolidation does exactly the latter as follows. The page metadata is consulted. For each subpage and each superblock of that subpage, if the superblock is in NM in swap mode, it will be swapped with the superblock in FM. If it is cached and compressed in NM, it will be decompressed and written back if the superblock is dirty or silently evicted if it is not dirty.

4 EXPERIMENTAL METHODOLOGY

This section provides the details of the experimental setup in Section 5.3.2, the benchmarks used in our evaluation in Section 4.2 and the models we use in our evaluation in Section 4.3.

4.1 Simulation Methodology and Parameters

We use Gem5 [4] based on the Simpoint methodology [10]. We run 10 representative slices of each application. We warm up caches before taking measurements in each slice for 100 million instructions and then replay the following 500 million instructions. We use workload mixes of eight benchmarks for eight cores in rate mode with the same benchmark run on every core.

We adopt the timing parameter configuration of HBM and DDR4 in Gem5, listed in Table 1. For the hybrid memory system, the capacity ratio between HBM and DDR4 is set to 1:4. We allocate

4-KB pages in an interleaved fashion to form congruence groups between HBM and DRAM according to Section 2.1 with five consecutive pages mapped so that the first one is mapped to HBM and the next four pages are mapped to DRAM. Each entry (64 bytes) in the metadata cache contains 16 consecutive metadata entries. Thus, the timing parameter of a 256-KB metadata cache is estimated in CACTI the same way as a classic 16-KB cache. Table 1 shows the detailed architectural parameters used.

Table 1: Simulator configuration.

Cores	8 cores, out-of-order, ARM, 3.2GHz
L1 Cache	Private, 64KB, 4-way, 4 cycle access latency
L2 Cache	Private, 512KB, 8-way, 12 cycle access latency
L3 Cache	Shared, 8MB, 8-way, 20 cycle access latency
Metadata Cache	512 sets, 8-way, 16 entry per line, 5-cycle latency
Compressor	CPack, 13-cycle compression latency, 8-cycle decompression latency [5]
Near Memory	HBM2, 1GHz, 2GB, 8 128-bit channels, 8 banks, tCAS=28ns, tRCD=12ns, tCL=18ns, tRP=14ns
Far Memory	DDR4-2400, 8GB, 64 bit channel, 8 banks, tCAS=32ns, tRCD=14.16ns, tCL=14.16ns, tRP=14.16ns

4.2 Benchmarks

We simulate all of the SPEC2017 benchmarks except the following:

- `roms` and `omnetpp` are excluded because they do not run properly on Gem5;
- `imagick, leela, povray`, and `exchange2` are excluded because of the small memory footprint (< 10MB) making them unsuitable for this study;
- `nab, namd, deepsjeng, perlbench, parest`, and `bwaves` are excluded because of too low LLC Misses-Per-Kilo-Instruction (MPKI) (MPKI < 1). However, we show results for `deepsjeng` and `bwaves` to represent this group of applications to show their impact on HMComp.

The compression algorithm used is CPack [5] with the compression and decompression latencies shown in Table 1. Table 2 shows the LLC MPKI statistics for a single core, the average compression ratio of blocks (64 bytes) in memory and memory footprint for 8 cores.

4.3 Simulation Models

We compare HMComp with a baseline configuration with HBM in flat mode without swapping (BL1), a baseline with swapping (BL2) and the two closest state-of-the-art proposals: Hybrid^2 [33] and Baryon [20]. The sizes of pages, subpages, superblocks and blocks are by default 4KB, 2KB, 512B and 64B, respectively, although we will present a sensitivity analysis with respect to the size of subpages and superblocks in Section 5.3.

- **BL1**. The baseline hybrid memory system according to Section 2.1.

Table 2: Benchmark characteristics.

Programs	MPKI	Compression Ratio	Memory Footprint
`mcf`	5.5	2.13	5.10 GB
`blender`	4.86	1.33	5.02 GB
`xalancbmk`	4.67	1.52	2.00 GB
`gcc`	4.32	3.22	7.68 GB
`lbm`	4.15	1.33	3.20 GB
`fotonik3d`	2.95	3.76	7.22 GB
`cactuBSSN`	2.27	1.16	6.18 GB
`wrf`	1.11	2.28	1.76 GB
`bwaves`	0.87	1.28	6.89 GB
`deepsjeng`	0.07	2.46	5.46 GB

- **BL2**. In BL2, superblocks referenced frequently in FM, according to the HMComp methodology in Section 3, will be swapped with superblocks in the same congruence group in NM.
- **Hybrid^2**. For Hybrid^2, we statically allocate a NM cache of 64 MB as proposed in Hybrid^2 [33]. We model it by modifying HMComp according to Section 3 to fix the cache size to 64 MB after the initialization process. Hybrid^2 will then decide whether to write back data or swap them to create the available NM cache space until the cache becomes full. To simplify the implementation of Hybrid^2, we do not save space in NM for the remap table but instead use the metadata cache provided by HMComp. This will give our implementation of Hybrid^2 a performance advantage over the original proposal[33].
- **Baryon**. Baryon [20] is modeled as Hybrid^2 with the difference that the size of the NM cache is not limited to 64MB but is instead 64 MB times the compression ratio using the CPack compression algorithm of the benchmark being modelled (see Table 2).

5 EXPERIMENTAL RESULTS

In this section, we present the results of the evaluation of HMComp. Section 5.1 evaluates the impact of HMComp on performance compared with the baselines (BL1 and BL2) and the closest state-of-the-art models: Hybrid^2 and Baryon. Section 5.2 evaluates the impact of HMComp on the FM traffic and Section 5.3 presents a sensitivity analysis of selected architectural parameters.

5.1 Speedup of HMComp over Other Models

Figure 9 shows the improvement of Instructions Per Cycle of BL2, Hybrid^2 and Baryon normalized to BL1. We can see that HMComp achieves the best performance with an average speedup of 24.0%, 13.4%, 7.7%, and 4.1%, compared to BL1, BL2, Hybrid^2 and Baryon, respectively. In Figure 9 benchmarks are sorted according to their MPKI as shown in Table 2 with the highest MPKI to the lowest MPKI, from left to right.

To understand the results for each individual benchmark, Figure 10 depicts the average latency ratio of FM accesses to NM accesses assuming BL1 and Figure 11 displays the ratio of FM superblock request-hits in the cache/compress mode in NM. A higher average latency ratio suggests a greater potential for performance

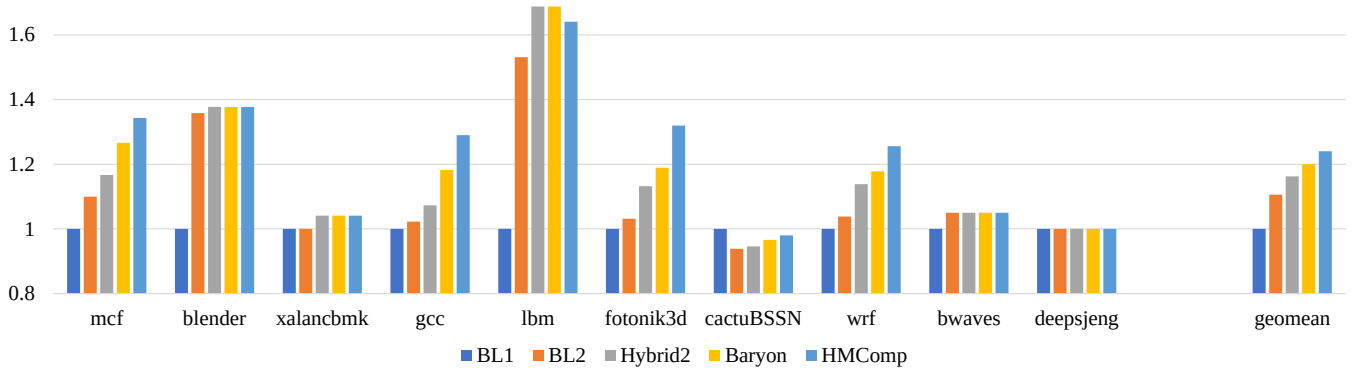


Figure 9: Geomean speedup of BL2, Hybrid^2, Baryon and HMComp normalized to BL1.

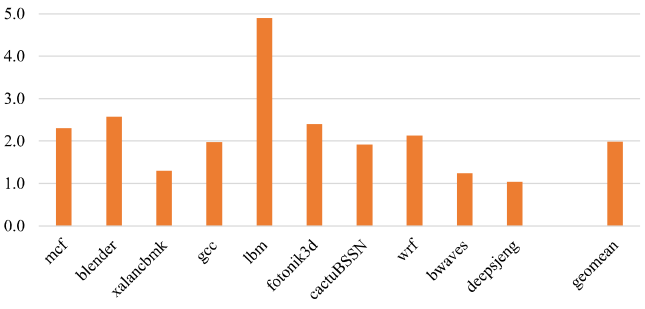


Figure 10: FM average memory access latency ratio, normalized to NM latency in BL1.

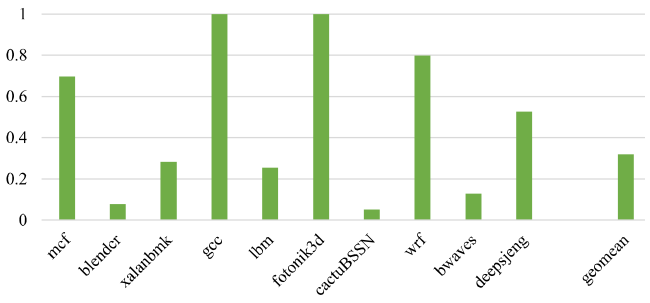


Figure 11: Fraction of FM superblock request that hit NM superblocks in cache/compress mode.

enhancement through the caching or swapping of FM data into NM. First we can see that HMComp shows a performance advantage over Hybrid^2 and Baryon for `mcf`, `gcc`, `fotonik` and `wrf` (see Figure 9). As we can see in Figure 11, the fraction of FM superblock accesses that hit in NM in cache/compress mode is about 70% for `mcf`, close to 100% for `gcc` and `fotonik` and 80% for `wrf`. The reason for the high fraction of accesses to NM is attributed to the high compression ratio for these benchmarks. As shown in Table 2, the average compression ratio per block is above 2× for the four benchmarks; 2.13, 3.22, 3.76 and 2.28× for `mcf`, `gcc`, `fotonik` and `wrf`, respectively. This would not lead to any performance advantage for HMComp unless there is a significant latency gap between FM and NM due to the higher bandwidth provided by NM (HBM). As we can see in Figure 10, the average latency ratio of FM accesses to NM accesses is more than two times for the same benchmarks; it is 2.2, 2.0, 2.4 and 2.1× for `mcf`, `gcc`, `fotonik` and `wrf`, respectively.

For `blender`, `xalancbmk` and `bwaves`, HMComp performs similarly with Hybrid^2 and Baryon and offers a slight performance advantage over the two baselines: BL1 and BL2. As can be seen in Table 2, the average per-block compression ratio is quite good – 1.33, 1.52 and 1.28× for `blender`, `xalancbmk` and `bwaves`, respectively – although not as high as for the first set of benchmarks. However, the fraction of FM accesses that hit in NM is substantially lower (5%, 28% and 15% for `blender`, `xalancbmk` and `bwaves`, respectively) which explains why the NM caches in Hybrid^2, Baryon and HMComp are not as effective and do not yield a noticeable performance advantage over BL2.

As for `deepsjeng`, we can see from Table 2 that it compresses quite well with a compression ratio of 2.46×. Moreover, as can be seen from Figure 11, this translates into a NM hit ratio of 55%. Unfortunately, the average latency ratio for FM accesses to NM accesses is close to one which takes away any performance advantage of BL2, Hybrid^2, Baryon and HMComp over BL1. The reason is that `deepsjeng` has a very low MPKI (0.07 according to Table 2).

For `lbm`, as we can see in Figure 9, HMComp performs slighly worse than Hybrid^2 and Baryon although it performs better than BL2 and BL1. From Table 2, we can see that `lbm` offers a compression ratio that is quite low (1.33 ×). In addition, the fraction of accesses that hit in NM (see Figure 11) is only about 20%. The statically assigned caches in Hybrid^2 and Baryon give them a slight performance advantage over HMComp. However, it is possible to conceive a system that like Hybrid^2 statically assigns a part of NM as a cache and operates the flat part of Hybrid^2 like HMComp. Such a system would get the advantages of Hybrid^2 as well as HMComp. We leave it for future work to evaluate such a system.

Finally, for `cactuBSSN` we see in Figure 9, that its performance is slightly worse for all system models than for BL1. In Figure 12, we collect statistics of the fraction of accesses to superblocks (y axis) accessed a certain number of times (x axis) for `blender`, `xalancbmk`,

lbm and cactuBSSN. We show these statistics for the selected applications because they all exhibit high traffic for swap operations according to Figure 11. Here, we can see that the hit rate in NM is very low (less than 30%). Figure 12 shows that all superblocks in cactuBSSN are accessed only 8 times whereas at least 30% of the superblocks for the other benchmarks are accessed more than 16 times.

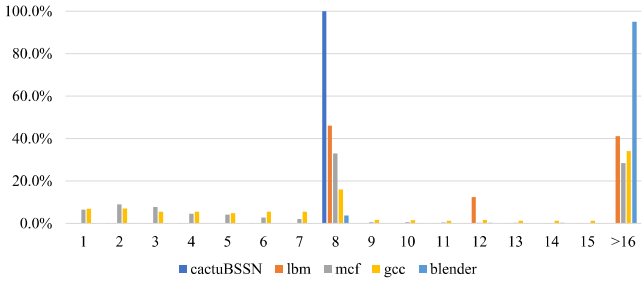


Figure 12: Histogram of access counters for 512-B superblocks.

5.2 Impact on Swap Traffic

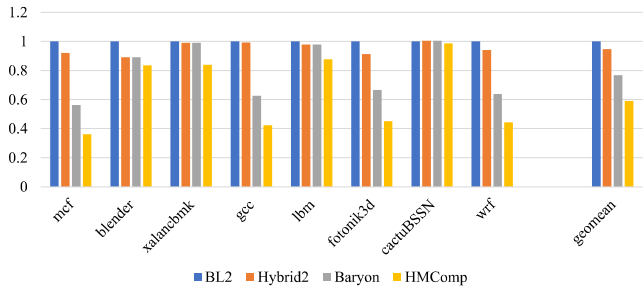


Figure 13: Swap traffic for Hybrid^2, Baryon and HMComp relative to BL2.

Figure 13 shows the traffic related to swap operations between FM and NM for Hybrid^2, Baryon and HMComp relative to BL2 for each benchmark (the lower the better). Overall, HMComp manages to cut the swap traffic by 41%, on average, compared to BL2. This should be contrasted by only 5% and 23% lower traffic for Hybrid^2 and Baryon, respectively, compared to BL2.

The reason for the slight reduction of traffic for Hybrid^2 is that it has a limited NM cache of only 64 MB. In contrast, Baryon and HMComp take advantage of the high compression ratio of some of the benchmarks to yield a substantially larger cache. Especially, we can see that for mcf, gcc, wrf, and fotonik3d, where the compression ratio ranges between 2.1 and 3.2× according to Table 2, the traffic reduction is substantial. For Baryon, the traffic reduction is about 40% for these benchmarks, whereas the traffic reduction for the same benchmarks under HMComp is as much as 60%.

5.3 Sensitivity Analysis

This section presents sensitivity analysis of the performance results with respect to the ratio of the amount of DRAM to HBM in Section 5.3.1, the metadata cache size in Section 5.3.2 and the superblock granularity in Section 5.3.3.

5.3.1 Impact of Ratio of DRAM to HBM. So far, we have assumed a memory configuration of 2-GB HBM and 8-GB of DRAM. To explore the sensitivity of performance for memory configurations, we also consider memory configurations with 1-GB HBM & 8-GB DRAM and 4-GB HBM & 8-GB DRAM. If we decrease the amount of NM (HBM) capacity we would expect a more severe bandwidth bottleneck problem in FM. In this analysis we exclude bwaves and deepsjeng because their MPKI are less than one (see Table 2).

Figure14 shows the IPC improvement for the two configurations relative to BL1: 1-GB HBM & 8-GB DRAM to the left and 4-GB HBM & 8-GB DRAM to the right. As can be seen in Figure14, when we consider the configuration with 1-GB HBM & 8-GB DRAM, performance of HMComp improves by 39.2%, while Baryon enjoys an increase of 34.9% and Hybrid2 experiences a 28.3% improvement. For the configuration with 4-GB HBM & 8-GB DRAM, the performance improvements are as expected lower than for the default configuration. HMComp shows a 22.0% improvement compared to the BL1, while Baryon and Hybrid2 demonstrate improvements of 18.8% and 16.1%, respectively. Overall, the results are consistent with the default configuration.

5.3.2 Impact of Metadata Cache Size. So far, we have assumed a metadata cache of 256 KB. Here, we explore a range of metadata cache sizes: from 32KB to 2MB. We have established the access time for the various cache sizes using CACTI [23]. These are shown in Table 3. As we discuss in Section , considering that each entry (64 bytes) in the metadata cache contains 16 metadata entries (32 bits), the timing parameter of a 256-KB metadata cache is configured as a conventional 16-KB cache in CACTI.

Table 3: MetaData Cache Timing Parameter.

Size (Bytes)	32K	64K	128K	256K	512K	1M
Latency (ns)	0.47	0.47	1.31	1.37	1.73	1.78

Figure 15 shows the IPC, taking misses to FM into account (where the metadata table is stored), normalized to a metadata cache of 32KB. As we would expect, the speedup improves up to a certain cache size and then drops for some of the benchmarks (e.g., fotonik3d) because of longer cache hit time. Considering the geometric mean of the speedup, we can see that performance peaks at 256 KB. Hence, a 256-KB metadata cache seems to be the best choice.

5.3.3 Impact of Superblock Granularity. The granularity of superblocks needs to strike a balance between prefetching coverage and accuracy. A too small a superblock will not capture the spatial locality and too large a superblock would fetch useless data. Additionally, the size of the metadata is also determined by the granularity of the superblock. The larger the superblock, the less metadata is needed. In order to establish the best superblock granularity, we conduct a sensitivity analysis with respect to its size. The subpage size is kept fixed at 2KB consistent with proposals in prior art [17, 20, 30, 31, 33]. The performance of HMComp as normalized to BL1 is depicted in Figure 16. deepsjeng and bwaves are not shown, since they have a low MPKI, according to Table 2.

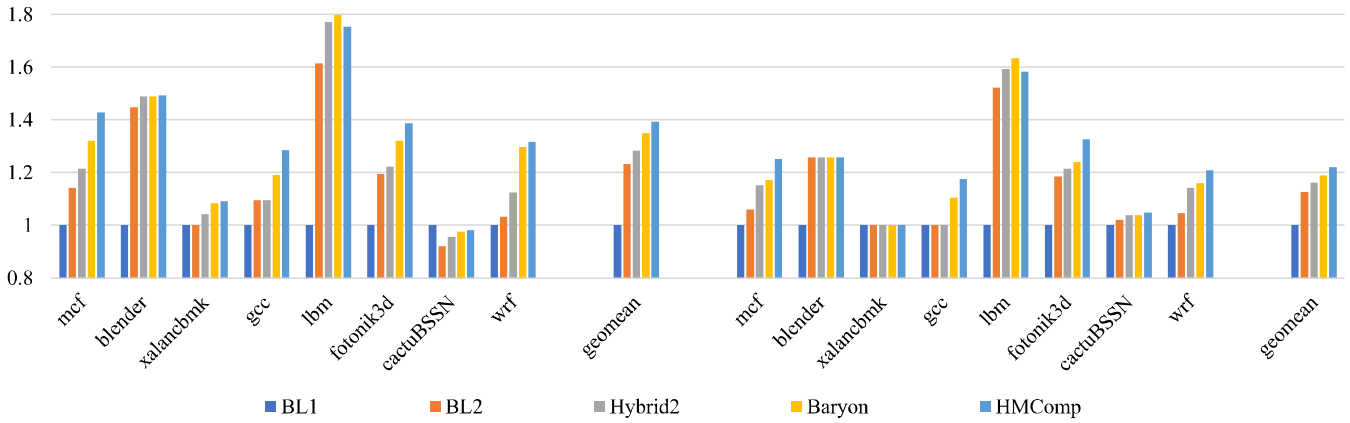


Figure 14: Performance improvement of two memory configurations relative to BL1. The left half shows a configuration with 1-GB HBM and 8-GB DRAM while the right half shows a confguration with 4-GB HBM and 8-GB DRAM.

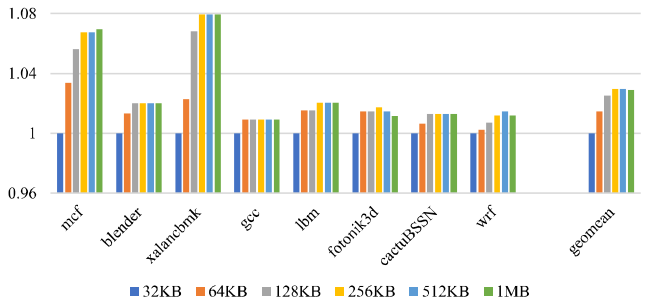


Figure 15: Performance comparison with different metadata cache sizes.

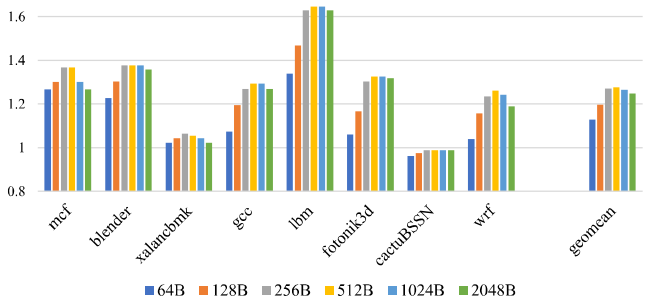


Figure 16: IPC improvements relative to BL1 versus size of superblock ranging from 64B to 2048B.

As shown in Figure 16, the performance improvement of HMComp, compared with BL1 for each core, increases from 12.8% to 26.4% and 27.1% when the superblock size is increased from 64B to 256B and 512B. However, when the size of the superblock is further increased to 1KB and 2KB, the performance of HMComp deteriorates due to the limited spatial locality. Consequently, a 512-B granularity of the superblock is a good tradeoff.

6 CONCLUSIONS

This paper proposes Hybrid Memory Compression (HMComp). Unlike previous work, HMComp exposes the *entire* near memory (NM) plus far memory (FM) capacity in flat mode to the system while dynamically exposing a cache in NM from capacity made available from compressing data in NM. The freed-up cache is used to bring more bandwidth-demanding FM data into NM to avoid costly swap operations. Through its novel management along with its metadata layout, metadata can be kept in FM. HMComp imposes virtually no area overhead in NM for metadata or for staging.

HMComp unlocks space for caching by selectively compressing data in NM. This is done by dynamically gauging compressibility and bandwidth demands of fine-grain access units in FM. By allowing fine-grain management of hybrid memory, HMComp carefully manages compressed blocks in HBM with a minimum of metadata needed. For example, it compresses HBM blocks where they originally are mapped and uses surplus ECC bits (in HBM) in a clever way to locate a compressed block and eliminates the use of remap tables in NM altogether.

Apart from presenting the detailed design of HMComp, this paper evaluates its performance compared to state-of-the-art hybrid memory schemes. The evaluation shows that HMComp offers a speedup of single-thread performance of up to 22% and on average 13% and a swap traffic reduction of up to 60% and by 41% on average compared to flat hybrid memory designs. Finally, we show that a quite modest metadata cache of 256 KB suffices to host the metadata cached from FM.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their valuable comments. This work is supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP),funded by Knut and Alice Wallenberg Foundation. 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.

REFERENCES

[1] Alaa Alameldeen and David Wood. 2004. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. (01 2004).

[2] Alexandra Angerd, Angelos Arelakis, Vasilis Spiliopoulos, Erik Sintorn, and Per Stenström. 2022. GBDI: Going Beyond Base-Delta-Immediate Compression with Global Bases. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1115–1127. https://doi.org/10.1109/HPCA53966.2022.00085

[3] Angelos Arelakis, Fredrik Dahlgren, and Per Stenstrom. 2015. HyComp: A hybrid cache compression method for selection of data-type-specific compression methods. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 38–49. https://doi.org/10.1145/2830772.2830823

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[5] Xi Chen, Lei Yang, Robert P. Dick, Li Shang, and Haris Lekatsas. 2010. C-Pack: A High-Performance Microprocessor Cache Compression Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 8 (2010), 1196–1208. https://doi.org/10.1109/TVLSI.2009.2020989

[6] C. Chou, A. Jaleel, and M. Qureshi. 2017. BATMAN: Techniques for maximizing system bandwidth of memory systems with stacked-DRAM. In *MEMSYS*. ACM, 268–280. https://doi.org/10.1145/3132402.3132404

[7] Chia Chen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2014. 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*. 1–12. https://doi.org/10.1109/MICRO.2014.63

[8] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. 2023. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *ASPLOS 2023* (Vancouver, BC, Canada). ACM, New York, NY, USA, 727–741. https://doi.org/10.1145/3582016.3582031

[9] Fazal Hameed and Jeronimo Castrillon. 2017. Rethinking on-chip DRAM cache for simultaneous performance and energy optimization. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 362–367. https://doi.org/10.23919/DATE.2017.7927017

[10] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. 2005. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism* 7, 4 (2005), 1–28.

[11] Mahzabeen Islam, Shashank Adavally, Marko Scrbak, and Krishna Kavi. 2020. On-the-Fly Page Migration and Address Reconciliation for Heterogeneous Memory Systems. 16, 1 (2020), 1–27. https://doi.org/10.1145/3364179

[12] JEDEC. 2022. JESD238 High Bandwidth Memory (HBM3) DRAM. (2022).

[13] Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 25–37. https://doi.org/10.1109/MICRO.2014.51

[14] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. 41, 3 (jun 2013), 404–415. https://doi.org/10.1145/2508148.2485957

[15] Jungrae Kim, Michael Sullivan, Esha Choukse, and Mattan Erez. 2016. Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 329–340. https://doi.org/10.1109/ISCA.2016.37

[16] Youngin Kim, Hyeonjin Kim, and William J. Song. 2023. NOMAD: Enabling Non-blocking OS-managed DRAM Cache via Tag-Data Decoupling. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 193–205. https://doi.org/10.1109/HPCA56546.2023.10071016

[17] Jagadish B. Kotra, Haibo Zhang, Alaa R. Alameldeen, Chris Wilkerson, and Mahmut T. Kandemir. 2018. CHAMELEON: A Dynamically Reconfigurable Heterogeneous Memory System. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 533–545. https://doi.org/10.1109/MICRO.2018.00050

[18] Jagadish B. Kotra, Haibo Zhang, Alaa R. Alameldeen, Chris Wilkerson, and Mahmut T. Kandemir. 2021. Dynamically Adapting Page Migration Policies Based on Applications' Memory Access Behaviors. In *ACM Journal on Emerging Technologies in Computing Systems*. 1–24. https://doi.org/10.1145/3444750

[19] Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W. Lee. 2015. A fully associative, tagless DRAM cache. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 211–222. https://doi.org/10.1145/2749469.2750383

[20] Yiwei Li and Mingyu Gao. 2023. Baryon: Efficient Hybrid Memory Management with Compression and Sub-Blocking. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 137–151. https://doi.org/10.1109/HPCA56546.2023.10071115

[21] Gabriel Loh and Mark D. Hill. 2012. Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap. *IEEE Micro* 32, 3 (2012), 70–78. https://doi.org/10.1109/MM.2012.25

[22] Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. 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)*. 126–136. https://doi.org/10.1109/HPCA.2015.7056027

[23] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.

[24] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry. 2012. Base-delta-immediate compression: Practical data compression for on-chip caches. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 377–388.

[25] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Pietro Cicotti, Erwin Laure, and Stefano Markidis. 2017. Exploring the Performance Benefit of Hybrid Memory System on HPC Environments. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 683–692. https://doi.org/10.1109/IPDPSW.2017.115

[26] Zhouxuan Peng, Dan Feng, Jianxi Chen, Jing Hu, and Chuang Huang. 2022. RHPM: Using Relative Hotness to Guide Page Migration for Hybrid Memory Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022), 1–1. https://doi.org/10.1109/TCAD.2022.3231836

[27] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M. Tullsen. 2017. 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)*. 433–444. https://doi.org/10.1109/HPCA.2017.39

[28] Moinuddin K. Qureshi and Gabe H. Loh. 2012. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 235–246. https://doi.org/10.1109/MICRO.2012.30

[29] Rambus. [n. d.]. Accelerating AI/ML applications in the data center with HBM3. ([n. d.]).

[30] Jee Ho Ryoo, Mitesh R. Meswani, Andreas Prodromou, and Lizy K. John. 2017. SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 349–360. https://doi.org/10.1109/HPCA.2017.20

[31] Jaewoong Sim, Alaa R. Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent Hardware Management of Stacked DRAM as Part of Memory. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 13–24. https://doi.org/10.1109/MICRO.2014.56

[32] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. 2019. LLC-Guided Data Migration in Hybrid Memory Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 932–942. https://doi.org/10.1109/IPDPS.2019.00101

[33] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. 2020. Hybrid2: Combining Caching and Migration in Hybrid Memory Systems. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 649–662. https://doi.org/10.1109/HPCA47549.2020.00059

[34] Carl A. Waldspurger. 2003. Memory Resource Management in VMware ESX Server. 36, SI (2003). https://doi.org/10.1145/844128.844146

[35] Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas. 2017. Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14.

Received 18 January 2024; revised 18 March 2024; accepted 16 April 2024