Bumblebee: A MemCache Design for Die-stacked and Off-chip Heterogeneous Memory Systems

Yifan Hua
Shanghai Jiao Tong University
Shanghai, China
huahuahuahua@sjtu.edu.cn

Shengan Zheng
MoE Key Lab of Artificial Intelligence, AI Institute
Shanghai Jiao Tong University
Shanghai, China
shengan@sjtu.edu.cn

Ji Yin
Shanghai Jiao Tong University
Shanghai, China
johnsmith_kyon@sjtu.edu.cn

Weidong Chen
Shanghai Jiao Tong University
Shanghai, China
weidong98@sjtu.edu.cn

Linpeng Huang
Shanghai Jiao Tong University
Shanghai, China
lphuang@sjtu.edu.cn

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10248000

***Abstract*—Emerging die-stacked memories can provide higher bandwidth than traditional off-chip DRAM and serve as an off-chip DRAM cache or part of OS-visible memory (POM). This paper presents Bumblebee, a new hybrid memory architecture combining the advantages of both DRAM cache and POM. The ratio of DRAM cache to POM is adjustable in real time to better exploit both temporal and spatial locality benefits for different memory access patterns. Our evaluations indicate at least 35.2% performance improvement and 10.9%∼20.1% less memory dynamic energy consumption for Bumblebee over state-of-the-art designs, as well as orders of magnitude less metadata storage space.**

***Index Terms*—Heterogeneous memory, Die-stacked high-bandwidth memory, Caching and migration**

I. Introduction

Many modern big-data applications have vast datasets that dwarf capacity-limited SRAM caches, resulting in excessive cache miss requests and severe bandwidth pressure to off-chip DRAM modules [4]. Consequently, the memory bandwidth becomes the bottleneck for these applications. Die-stacked or on-die memories (e.g., HBM [1], HMC [3]) have been proposed to offer substantially higher bandwidth and better energy efficiency than traditional off-chip DRAM. These high-bandwidth memories (abbreviated as HBM) have limited capacity and are often complemented with a larger off-chip DRAM.

Recent works have employed HBM as an off-chip DRAM cache [11]–[15], part of OS-visible memory (POM) [6]–[10], and both DRAM cache and POM (hybrid mode) [17], [18]. DRAM cache designs react fast to hotness changes by fetching all requested data, but take the HBM capacity away from the memory system and have a bad performance for workloads with weak temporal locality. POM designs enhance the OS-visible memory capacity and bandwidth efficiency, but make the migration decision slower since only data with potential for future reuse is migrated. Hybrid mode designs aim to combine the advantages of both DRAM cache and POM.

Unfortunately, existing hybrid mode designs [17], [18] suffer from three limitations: (1) Incapability of supporting an adjustable ratio of DRAM cache to POM during runtime. Fixed DRAM cache and POM capacity lack the flexibility to match different memory access patterns. (2) Unnecessary migration overhead for mode switch between DRAM cache and POM stemming from the separate DRAM cache and POM space. For example, for evicting a page from DRAM cache to POM, a victim page in POM is swapped out to off-chip DRAM, and then cached for the subsequent access, which brings unnecessary migration cost. (3) Large metadata management and access overheads. Their space-inefficient metadata structures such as pointers to index pages and tags to manage DRAM cache lines consume large storage space. Besides, they employ small metadata management granularity to reduce over-fetching. Not only is the metadata size too large to be accommodated in on-chip SRAM, but also the potential spatial locality benefit is not fully exploited.

In this paper, we propose a new hybrid memory architecture named Bumblebee to preserve the advantages of both DRAM cache and POM while overcoming the above three limitations. All HBM can be utilized as DRAM cache (cHBM) or OS-visible POM (mHBM). HBM can be dynamically switched between cHBM and mHBM over time to better exploit both temporal and spatial locality benefits for different memory access patterns. cHBM can be compelled to become mHBM for high memory footprint condition to maximize the total system memory capacity and reduce page faults. Caching and migration decisions are made based on workloads' temporal and spatial locality features as well as the system memory footprint. Bumblebee employs a unified set-associative mechanism for PRT (Page Location Entry Remapping Table) to track page migration and BLE (Block Location Entry) array to record blocks' information in an HBM page for evaluating the spatial locality. A hot table is designed to track data hotness changes and provide the temporal locality information. To minimize the migration cost between cHBM and mHBM, all cHBM and mHBM space are not separate but multiplexed to enable the mode switch process to move only necessary data. To limit the metadata storage space, space-efficient metadata structures and suitable cHBM and mHBM management granularity are employed. The over-fetching risk is lowered by providing more cHBM capacity for workloads with weak spatial locality to reduce the eviction frequency, extending pages' residency in mHBM, and preventing data with a low access frequency into HBM. This paper makes the following contributions:

- We present Bumblebee, a new hybrid memory architecture in which HBM may serve as both cHBM and mHBM. The ratio of cHBM to mHBM is adjustable in real time to better exploit both temporal and spatial locality benefits for different memory access patterns, without rebooting.
- All cHBM and mHBM space in Bumblebee are not separate but multiplexed, which minimizes the data movement overhead for mode switch between cHBM and mHBM.
- Bumblebee greatly reduces the storage space for metadata by 1∼2 orders of magnitude through employing space-efficient metadata

* Linpeng Huang and Shengan Zheng are corresponding authors.

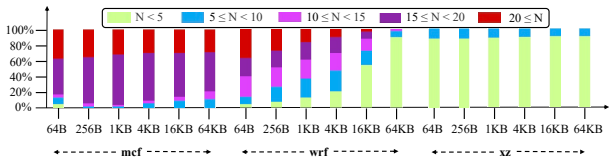


Fig. 1. Percentage of cache lines with different access numbers before eviction in 1GB cHBM. N represents the average access number for each 64B data in different sizes of cache lines.

structures and appropriate migration and caching granularity with a low over-fetching risk.

• In our evaluations, Bumblebee outperforms state-of-the-art designs by at least 35.2%, incurs 17.9% less HBM traffic and 9.1% less off-chip DRAM traffic than the best respectively, and consumes 10.9%∼20.1% less memory dynamic energy.

II. Background and Motivation

In order to break the memory bandwidth wall, many designs have been proposed to combine HBM with traditional off-chip DRAM in hybrid memory systems [5]–[18]. State-of-the-art designs leverage the on-chip HBM as an off-chip DRAM cache (cHBM), POM (mHBM), and both DRAM cache and POM (hybrid mode).

A. Three types of HBM utilization

HBM used as cHBM: A large body of works have utilized HBM as cHBM between the last level cache (LLC) and off-chip DRAM. They can be divided into two classes: block-based [11]–[13] and page-based [14]–[16]. To enhance the caching capacity and better exploit the temporal locality, common block-based cHBM manages data at 64B cache line granularity. Unfortunately, tags may occupy 12.5% of the HBM capacity [12]. Besides, block-based cHBM has a poor hit rate for workloads with weak temporal and strong spatial locality [13]. Page-based cHBM reduces the tag overhead by caching 1∼8KB pages. However, many pages contain data that is not accessed prior to the page's eviction from the cHBM, wasting memory bandwidth [14]. In general, small cache lines better exploit the cache but have higher tag overhead. Large cache lines reduce the tag overhead but may cause over-fetching.

HBM used as mHBM: Contrary to cHBM designs, mHBM designs make all HBM capacity visible to OS and have the potential to utilize the bandwidth of all memories for serving memory requests. As a result, they can potentially reap the benefits of both higher aggregate bandwidth and capacity. However, the high remapping overhead [5]–[7] and over-fetching issue caused by coarse migration granularity [8]–[10] still plague modern mHBM designs.

HBM in hybrid mode: Some researches aim to combine the advantages of both cHBM and mHBM, and they show that hybrid mode could gain more performance benefits than single mode. State-of-the-art hybrid mode designs adopt statically reconfigurable mechanisms to manage the two modes. In particular, KNL [18] supports 25% or 50% HBM as cHBM and Hybrid2 [17] fixes a small cHBM capacity of 64MB or 128MB. The other HBM is used as mHBM. Both of them require a system reboot to switch from one hybrid configuration to another. They employ small metadata granularity for HBM management and these metadata cannot be totally accommodated in SRAM. Hundreds of kilobytes SRAM is used as a metadata cache to place those hot pages' information.

B. Motivation

As described before, state-of-the-art hybrid mode designs fix the cHBM and mHBM capacity, which cannot always meet the memory requirements for different memory access patterns. Figure 1 shows the memory access patterns of three representative SPEC2017 [20] workloads' slices selected by Simpoint [19] as an example: we collect the average access number for each 64B data in different sizes of cache lines before eviction in 1GB cHBM. For the slice of mcf (strong spatial and strong temporal locality), both the large and small cache lines achieve a high access number. To reduce the high tag overhead and better leverage the spatial locality and memory bandwidth, most HBM is preferable to be used as mHBM with large management granularity without causing over-fetching. For the slice of wrf (weak spatial and strong temporal locality), with the cache line size increasing, the number of hot cache lines decreases, which means large cache lines may bring over-fetching. Thus, a small part of HBM is better to serve as mHBM to store those large hot data while the other HBM is recommended to be utilized as cHBM with small management granularity. For the slice of xz (strong spatial and weak temporal locality), most data is rarely accessed and hard to benefit from caching. A large amount of data movement may generate significant bandwidth cost and frequent hot data evictions. Thus, most HBM is favored as mHBM with large management granularity for better bandwidth efficiency and spatial locality, with a non-aggressive migration scheme. To sum up, for different kinds of workloads, a fixed cHBM capacity cannot fully exploit the temporal and spatial locality and utilize all the memory bandwidth. Worse still, they may cause over-fetching and bring unnecessary data movement cost. *The ratio of cHBM to mHBM should be dynamically adjustable over time.*

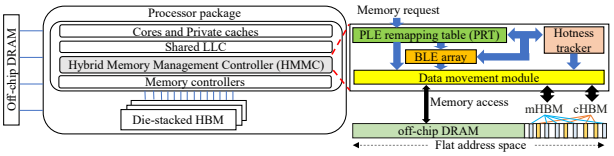


Fig. 2. System overview.

In addition, the statically reconfigurable designs consume more bandwidth for data movement between cHBM and mHBM due to the separate cHBM and mHBM space. For example, for evicting a page from cHBM to mHBM, another mHBM page as a victim is swapped out to off-chip DRAM, and then cached for the subsequent access, which brings extra unnecessary migration costs. Thus, *the space of cHBM and mHBM need to be multiplexed in order to minimize the data movement overhead for switching the two modes.*

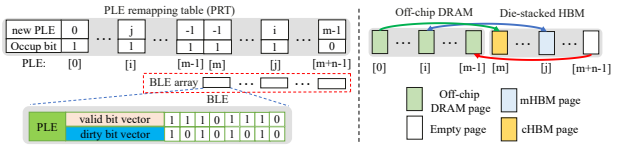
Besides, the metadata access latency (MAL) on the critical path in modern hybrid mode designs cannot be ignored. They employ small caching and migration granularity to lower the over-fetching risk. Not only is the potential spatial locality not fully exploited, but also the metadata storage space can reach tens of megabytes, which cannot be accommodated in on-chip SRAM and takes large HBM capacity away from the memory system. Their space-inefficient metadata structures such as pointers to index pages and tags to manage DRAM cache lines consume large storage space as well. We calculate the MAL in HBM and find that it accounts for $2\% \sim 26\%$ of the total memory request latency for workloads in section IV. As a result, *the metadata size should be minimized as small as possible.*

III. Bumblebee Architecture

Bumblebee is a hybrid memory architecture in which HBM can be used as either cHBM or mHBM. Data can be fetched from off-chip DRAM to cHBM in block size, and migrated in page size between off-chip DRAM and mHBM. The ratio of cHBM to mHBM is adjustable in real time to match different memory access patterns.

A. Bumblebee system overview

Figure 2 presents the overall architecture of our system. A Hybrid Memory Management Controller (HMMC) is added between the



(a) The PRT and BLE array. New PLE: the page's actual location (-1: has not been allocated). Occup bit 1/0: the page space is occupied/not occupied.

(b) Data movement examples in a remapping set indicated by arrows.

Fig. 3. An example of the unified set-associative mechanism for caching and migration. The data movement indicated by arrows in (b) corresponds to the values of PRT and BLE array in (a).

shared LLC and the memory layer to facilitate the migration along with the DRAM cache functionality. Metadata in HMMC consists of three components: the PRT (PLE (Page Location Entry) remapping table), the BLE (Block Location Entry) array, and the hotness tracker. For each LLC miss memory request, the PRT is responsible for querying the actual location of a page, either in off-chip DRAM or mHBM. The BLE array records the information of blocks in HBM pages to manage cHBM and provides the spatial locality information. The hotness tracker keeps track of the most recently visited hot pages to provide the temporal locality information and monitors the memory footprint, determining the migration and caching logic in Bumblebee. The metadata is only a few hundred kilobytes in size and can be entirely placed in on-chip SRAM (described in section IV-B). Similar to previous works, the data movement module moves data asynchronously and ensures that those data being moved can be fetched by memory access requests. By using the HMMC, HBM is logically, not physically, partitioned between cHBM and mHBM.

For high memory footprint condition, in order to provide more OS-visible memory, part of cHBM may be compelled to become mHBM, but the remaining HBM can still serve as cHBM. For low memory footprint condition, since the data migration granularity of mHBM (page size) is larger than the fetching granularity of cHBM (block size), data with strong spatial locality is preferred to be placed into mHBM, which can ensure the hit rate of HBM, better utilize all the memory bandwidth, and decrease the metadata query overhead for blocks. cHBM is better suited to pages with weak spatial locality and only hot blocks need to be cached to reduce over-fetching. In this way, the whole HBM not only is visible as a flat address space for OS, but also can flexibly serve as cHBM or mHBM most of the time to match different workloads' memory access patterns.

B. Memory space layout and metadata

Since cHBM can be flexibly switched to mHBM and the metadata does not take up any memory space, all the off-chip DRAM and HBM are visible as a flat address space for OS, as shown in Figure 2. The space of cHBM and mHBM in Bumblebee are multiplexed. That means any HBM space can serve as either cHBM or mHBM, and the mode switch (in section III-E) between cHBM and mHBM only modifies the metadata and requires moving necessary blocks.

For metadata management, direct remapping has a bad performance [23] due to the poor flexibility, uneven HBM utilization and frequent page swaps. Besides, the hardware query overhead for a fully-associative remapping table is unacceptable on chip. Therefore, for a balance between the hardware overhead and performance, Bumblebee adopts a unified set-associative mechanism to manage metadata both for caching and migration, as Figure 3 illustrates. An off-chip DRAM page is only allowed to be cached or migrated to another HBM page location in the same set. Since the metadata query is on the critical path and large quantities of memory accesses may degrade the performance, all metadata, including the PRT, the BLE array, and the hotness tracker, are stored in on-chip SRAM.

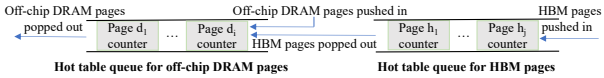


Fig. 4. The hot table in a remapping set.

PLE remapping table (PRT). The set-associative PRT holds the new PLEs and the Occup bits in each set, as Figure 3(a) shows. Each set includes m off-chip DRAM pages and n HBM pages. The PLE (Page Location Entry) refers to the original page index (i.e., offset from the first page in this set) in the remapping set, decided by the OS memory allocator and the virtual to physical address mapping mechanism in OS. The new PLE combines the functions of page address remapping (the remapped page index) and page allocation (-1: the page has not been allocated). For example, the blue arrow in Figure 3(b) indicates a swap between the ith and jth pages, recorded by the corresponding ith and jth new PLEs. One PLE requires less storage space ($\lceil \log_2 (m+n) \rceil$ bits) than a traditional tag or pointer (a few bytes). The Occup bit indicates if the memory page space has been occupied, queried by the page allocation process.

Block Location Entry (BLE) array. One BLE contains a PLE, a valid bit vector, and a dirty bit vector as Figure 3(a) shows. For a cHBM page, the corresponding BLE indicates if an off-chip DRAM page is cached in it and if the blocks in the page are valid and dirty. For an mHBM page, the valid bit vector records all accessed blocks in the page for evaluating the spatial locality.

Hotness tracker. In each remapping set, the hotness tracker includes a hot table and five parameters: the HBM occupied ratio (R_h), a hotness threshold (T) to decide if an off-chip DRAM page should be brought in HBM for high R_h condition, the number of cHBM pages (N_c), and the number of mHBM pages in which most blocks have/have not been accessed (N_a/N_n). To reduce the metadata storage overhead and get the temporal locality information, the hot table only monitors the hottest pages, including all HBM pages and the recently accessed off-chip DRAM pages. The hot table includes two queues with LRU replacement strategy, one for HBM pages and the other for off-chip DRAM pages, as shown in Figure 4. Each entry in the queue serves as a counter to record the access number for a page before popped out from the queue. The popped-out HBM page entries are pushed back into the off-chip DRAM queue and incur page evictions from HBM to off-chip DRAM. The five parameters are used for making data movement decision and detailed in section III-E.

C. Memory access flow

Figure 5 illustrates the memory access flow. The PRT miss (①) indicates that the requested page has never been allocated and needs to be assigned an address in PRT. After that, the memory request is sent to the allocated memory address. In the case of a PRT hit (②), the target page may reside in mHBM (③) or off-chip DRAM (④). For ③, the memory request goes to mHBM and for ④, HMMC checks the BLE array if the page is cached in cHBM. If the page is not cached (⑤), the memory request directly goes to off-chip DRAM and if the page is cached (⑥), HMMC further checks if the target block is cached as well. For block cached (⑦), the requested data is in cHBM and for block not cached (⑧), the memory request goes to off-chip DRAM. All memory accesses update the corresponding metadata and may incur data movement asynchronously.

D. Page allocation process

Few previous studies discuss the page allocation process. However, simply allocating all pages to off-chip DRAM or HBM by the OS memory allocator could not fully exploit all benefits of the hybrid mode design. A suitable page allocation mechanism could reduce

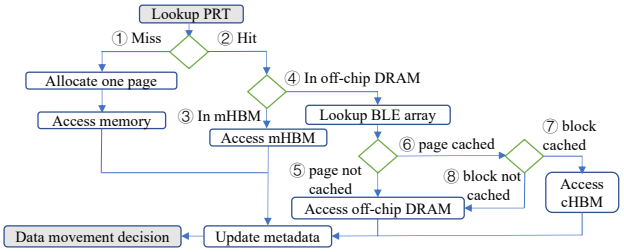


Fig. 5. Memory access path.

the migration cost for hot pages migrated from off-chip DRAM to HBM and cold pages evicted from HBM. For a PRT lookup miss, the requested page should be assigned an address in PRT. Since the PRT records the remapping information for all pages in both HBM and off-chip DRAM, the page to be allocated could be remapped to any free memory space. Based on the observation that adjacent allocation requests intend to have similar memory access patterns [24], we propose a hotness-based remapping allocation mechanism. If the recently allocated pages still reside in the hot table queue for HBM pages and there is free HBM space available, the page is allocated in HBM. Otherwise, the page should be allocated in off-chip DRAM.

E. Data movement decision

In this section, Bumblebee makes the data movement decision based on the temporal and spatial locality features of different memory access patterns and the system memory footprint to support flexible cHBM and mHBM capacity, lower the over-fetching risk, maximize the OS-visible memory capacity, and remove the eviction latency of cHBM pages from the critical path in the case of memory shortage. Memory accesses bring changes to the hotness tracker and as a result, incur data movement. Besides, for high memory footprint, data movement is triggered to meet OS memory requirements.

For spatial locality, the BLE array tracks the accessed blocks in both cHBM and mHBM pages. If most blocks in a page have been fetched into cHBM, that means the page has a strong spatial locality and should be switched to an mHBM page. More requested off-chip pages should be migrated to mHBM if most HBM pages have strong spatial locality. mHBM pages are switched from cHBM pages (most blocks in the page have been accessed) or migrated from off-chip DRAM (not sure if most blocks in the page have been accessed). Thus, mHBM pages with a high access ratio reflect a strong spatial locality degree while those with a low access ratio and the remaining cHBM pages reflect a weak spatial locality degree. The spatial locality degree (SL) in a remapping set is evaluated by:

$$SL = N_a - N_n - N_c \tag{1}$$

For $SL{>}0$ (strong spatial locality), more hot data should be brought in mHBM to better exploit the spatial locality and utilize the memory bandwidth. For $SL \leq 0$ (weak spatial locality), hot data should be cached in cHBM to reduce over-fetching.

For temporal locality, Bumblebee uses the hot table to track hot and recently accessed pages. It is hard to benefit from bringing data with low access frequency into HBM, which wastes memory bandwidth and may cause frequent hot page evictions for high HBM footprint condition, resulting in performance degradation. The threshold T in the hotness tracker can alleviate this issue. If R_h is high, for $SL{>}0$, only pages whose hotness value is larger than T are permitted to be migrated to mHBM and for $SL \leq 0$, only blocks in a page whose hotness value is larger than T are permitted to be cached in cHBM. In this way, for weak temporal locality workloads, a large amount of data with low access frequency is not brought into HBM, which reduces the data movement overhead and eviction frequency of hot pages, and better utilizes the off-chip DRAM bandwidth. For strong temporal locality workloads, despite some hot data that may benefit from caching and migration not brought in HBM, the eviction frequency of those hottest pages is reduced to make them serve more memory requests before eviction. The off-chip DRAM bandwidth is better utilized and the data movement overhead is reduced as well.

Based on the above analysis to better exploit both temporal and spatial locality benefits for different memory access patterns, the data movement triggered by memory access is shown as follows.

Data movement triggered by memory access. (1) For accessing an off-chip DRAM page, if $SL{>}0$ with a low R_h, the page is migrated to mHBM due to the strong spatial locality. If $SL{>}0$ with a high R_h, the page is migrated to mHBM only if its hotness value is larger than T. If $SL \leq 0$ with a low R_h, the page is cached to cHBM and only the requested block is fetched. If $SL \leq 0$ with a high R_h, the page is cached to cHBM only if its hotness value is larger than T. (2) For accessing a cHBM cached page, if the target block is not cached in cHBM, Bumblebee caches the block. If most blocks in the page have been cached, the cHBM page turns into an mHBM page. Only blocks not cached are fetched from off-chip DRAM, minimizing the data movement overhead for mode switch. (3) Accessing an mHBM page does not incur data movement.

Data movement triggered by high memory footprint to provide more free HBM space, evict zombie pages in HBM, and remove the eviction latency of cHBM pages from the critical path in the case of memory shortage is detailed below.

Data movement triggered by high memory footprint. (1) If a cHBM page entry is popped out from the hot table queue for HBM pages, the corresponding cHBM page is evicted to off-chip DRAM. (2) Since evicting an mHBM page to off-chip DRAM consumes much higher bandwidth (at least 2x) than a cHBM page, mHBM pages to be evicted have one more chance to reside in HBM. The mHBM page to be evicted is switched to cHBM mode without data migrated to off-chip DRAM and all blocks in the page are marked as dirty. The mode switch from mHBM to cHBM is designed as a buffering mechanism for increasing colder pages in mHBM to reduce the migration cost resulting from hotness fluctuations and extend the residency in HBM to serve more memory requests. There is no data movement cost for mHBM switched to cHBM due to our multiplexed space design. In this way, if these pages then become hotter, no data movement is required. (3) For high R_h, in case of both the head page in the hot table queue for HBM pages and its counter value remain unchanged over a long time, the page (zombie page) should be evicted to off-chip DRAM since there is no other page able to evict the zombie page from HBM. (4) If all memory in the set is used by OS, that means all the HBM in this set serves as mHBM. The five parameters are reused to track the hottest pages in off-chip DRAM and the coldest pages in HBM for swap. (5) If the system memory footprint is high in general (i.e., the address in LLC miss memory request packet is larger than the off-chip DRAM capacity), cHBM pages in multiple remapping sets are flushed to off-chip DRAM. This batching mechanism provides more OS-visible memory to reduce page faults. For pages to be allocated in these sets later, there is no need to wait for the eviction from cHBM to make free page space, which removes the eviction latency from the critical path. In these sets, all HBM will not be used as cHBM until the OS memory footprint drops.

To sum up, the migration and caching mechanism in Bumblebee could address the limitations of state-of-the-art hybrid mode designs for different kinds of workloads in Figure 1. For workloads with strong spatial and strong temporal locality, most HBM is used as

Table I. System configuration.

Core	ARM A72(AArch64), 3600MHz
IL1/DL1 cache	private 64 KB per core, 4-way, LRU
L2 Cache	private 256 KB per core, 8-way, SRRIP
L3 Cache	shared 8 MB, 16-way, DRRIP
HBM2 [1]	1 GB, 8 128-bit channels, 512B interleaved, 8 banks, tCAS-tRCD-tRP: 7-7-7, VDD:1.2, IDD0:65, IDD2P/N:28/40, IDD3P/N:40/55, IDD4W/R:500/390, IDD5/6:250/31
Off-chip DDR4_3200 [2]	10 GB, 2 64-bit channels, 8 banks, tCAS-tRCD-tRP: 22-22-22, VDD:1.2, IDD0:52, IDD2P/N:25/37, IDD3P/N:38/47, IDD4W/R:130/143, IDD5/6:250/30

Table II. Benchmark characteristics (MPKI: LLC Misses per Kilo Instructions).

	Benchmark	MPKI	Footprint(GB)
High MPKI	roms	31.9	10.6
	lbm	31.4	5.1
	bwaves	20.4	7.5
	wrf	18.5	2.7
Medium MPKI	xalancbmk	16.9	0.6
	mcf	16.1	0.2
	cam4	13.8	10.8
	cactuBSSN	12.2	2.9
Low MPKI	fotonik3d	2.0	0.2
	x264	0.9	1.9
	nab	0.8	0.9
	namd	0.5	1.9
	xz	0.4	7.2
	leela	0.1	0.1

mHBM for better spatial locality and memory bandwidth efficiency without over-fetching. For workloads with strong spatial and weak temporal locality, most HBM is used as mHBM with a non-aggressive migration scheme to better utilize all the memory bandwidth and exploit spatial locality benefits. Data with a low access frequency does not bring significant memory bandwidth waste and frequent evictions. For workloads with weak spatial and strong temporal locality, most HBM is used as cHBM to better exploit the temporal locality benefit and reduce over-fetching. Compared to the fixed cHBM capacity in existing hybrid mode designs, more cHBM capacity contributes to a lower eviction frequency for data in cHBM. For workloads with weak spatial and weak temporal locality, few data are moved.

IV. Evaluation

A. Experimental setup

Table I gives our system configurations. We use the gem5 simulator [21] and DRAMSim2 [22] to model Bumblebee, compared against the latest hybrid mode design: Hybrid2 [17], POM design: Chameleon [5] and three cache designs: Banshee [16], Alloy Cache (AC) [12] and Unison Cache (UC) [15]. We allow 512KB SRAM to store metadata for all designs for a fair comparison. The configurations for Bumblebee are shown as follows. Both cHBM and mHBM are managed with 8-way associativity. The hot table monitors the eight recently accessed off-chip DRAM pages for a balance between the performance and metadata size. We set T as the smallest hotness value of HBM pages in each set to match the temporal hotness pattern. The R_h is defined as high if its value reaches 1 to maximize the HBM utilization rate. We use the SPEC2017 benchmarks [20] in Table II to evaluate our design. We simulate at least 6 billion instructions for each benchmark using the Simpoint [19]. All our results are normalized to a baseline system without HBM.

B. Design space exploration and over-fetching analysis.

Bumblebee can be configured with any block and page size, which affects the performance and metadata size. To place all metadata in SRAM, we limit the metadata size within 512KB and show some possible configurations in Figure 6. We use the average normalized Instruction Per Cycle (IPC) for all benchmarks in Table II during execution as a performance metric for the speedup comparison. Smaller blocks miss the opportunity to exploit spatial locality while larger blocks cause over-fetching. Thus, a block of 2KB is a good compromise between the spatial locality exploitation and bandwidth consumption. For the same block size, 64KB pages reduce the over-fetching and perform better than 96KB and 128KB pages. Our design achieves the best performance at 2KB blocks and 64KB pages, and for the rest of experiments in this paper, we present our results in this configuration. The metadata storage space (334KB: 110KB PRT, 136KB BLE array, and 88KB hotness tracker) is reduced by 1∼2 orders of magnitude compared to prior designs.

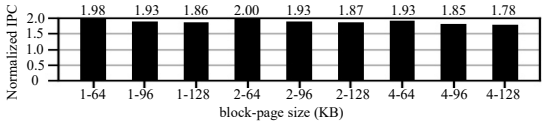


Fig. 6. Normalized IPC speedup for different configurations in Bumblebee.

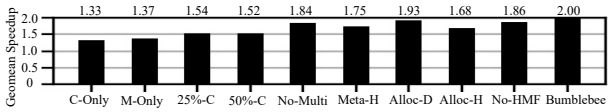
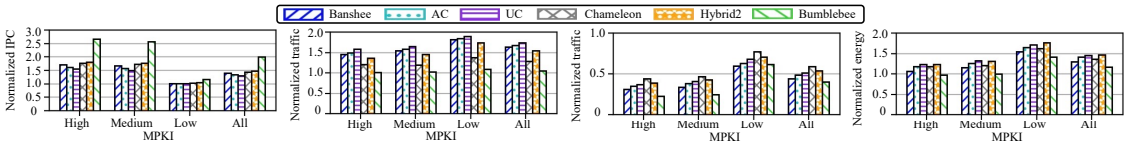


Fig. 7. Performance factors breakdown.

The sizes of blocks and pages in Bumblebee are larger than those in Hybrid2 (256B blocks and 2KB pages). We analyze the over-fetching by collecting the percentage of data brought in HBM but unused: 13.7% for Hybrid2 and 13.3% for Bumblebee. The reasons for the over-fetching of large blocks and pages reduced by Bumblebee are as follows: (1) The over-fetching is mainly caused by workloads with weak spatial locality. The adjustable design in Bumblebee can provide more cHBM capacity (up to 1GB) than Hybrid2 (64MB) for workloads with weak spatial locality, which greatly reduces the eviction frequency and enables large blocks in cHBM to serve more memory requests before eviction. (2) Our data movement mechanism only permits data in a page that reaches a certain hotness level (i.e., T) to be brought in HBM for high memory footprint condition, which prevents pages with low hotness into HBM and reduces the eviction frequency of pages in HBM. Hybrid2 brings all requested blocks into cHBM, which causes significant over-fetching for workloads with weak spatial locality. (3) The buffering mechanism for mHBM page eviction extends the pages' residency in HBM, enabling data brought in HBM to serve more memory requests before eviction.

C. Performance breakdown

The performance speedup of Bumblebee can be attributed to the cHBM and mHBM combination, the dynamically adjustable cHBM and mHBM capacity, the multiplexed cHBM and mHBM space, the metadata access overhead reduction, the allocation and the data movement mechanism. Figure 7 illustrates the average normalized IPC speedup for all SPEC2017 benchmarks in Table II to show the effect of each factor. From left to right: C-Only and M-Only represent all the HBM used as cHBM and mHBM, respectively. The more benefits achieved by M-Only than C-Only mainly stem from the better memory bandwidth efficiency for both HBM and off-chip DRAM. We also evaluate the performance of fixing the cHBM capacity of 25% and 50% total HBM capacity (25%-C and 50%-C) and both of them outperform the single mode designs. A hybrid mode design without the multiplexed cHBM and mHBM space (No-Multi) brings more data movement overhead for mode switch, wasting both HBM



(a) Normalized IPC speedup. (b) Normalized HBM traffic. (c) Normalized off-chip DRAM traffic. (d) Normalized memory dynamic energy.

Fig. 8. Comparing Bumblebee against state-of-the-art designs.

and off-chip DRAM bandwidth. Placing all the metadata in HBM (Meta-H) degrades the performance mainly due to the performance gap between SRAM and HBM as well as more HBM traffic. Alloc-D and Alloc-H represent allocating all pages to off-chip DRAM and HBM respectively. Compared to our hotness-based remapping allocation, Alloc-D consumes more bandwidth for hot data migration from off-chip DRAM to HBM. Alloc-H reduces the migration cost for workloads with low memory footprint since all the data can be placed in HBM while incurs significant bandwidth waste for high memory footprint workloads due to a large amount of data evicted from HBM. Without the data movement triggered by high memory footprint (No-HMF), simply evicting HBM pages popped out from the hot table queue degrades the system performance due to the extra eviction overhead for mHBM pages, the data movement hindered by zombie pages to react to the hotness changes, and the eviction latency of cHBM pages on the critical path in the case of memory shortage.

D. Performance

With the same system configuration, Bumblebee outperforms state-of-the-art designs by at least 46.7%, 44.9%, 9.9% and 35.2% on average for high, medium, low and all MPKI benchmarks respectively, as shown in Figure 8(a). Compared to Hybrid2, Bumblebee makes better use of HBM due to the adjustable capacity for cHBM and mHBM. Besides, since the metadata storage space is greatly minimized and the cHBM and mHBM space are multiplexed, the overhead for metadata access and mode switch are reduced by 69.7% and 44.6% on average respectively. Chameleon is designed based on POM [6] with the added option to economize on migration bandwidth. It restricts only one HBM sector in each remapping set, which brings uneven HBM utilization rate in different sets and frequent sector migration cost. The metadata access overhead in HBM degrades Chameleon's performance as well. Bumblebee outperforms cache designs by better exploiting the spatial locality benefit, improving the memory bandwidth efficiency, reducing the metadata management overhead and lowering the eviction frequency for hot data.

E. Memory traffic analysis and energy consumption

Figure 8(b) and Figure 8(c) illustrate the normalized HBM and off-chip DRAM traffic for each benchmark group. Bumblebee economizes bandwidth for both HBM (17.9% less traffic than the best, i.e., Chameleon) and off-chip DRAM (9.1% lower than the best, i.e., Banshee), which is one of the reasons why Bumblebee has the best performance. The traffic reduction mainly attributes to lowering the eviction frequency for pages in cHBM and mHBM, preventing data with a low access frequency into HBM, lowering the over-fetching risk, eliminating the metadata query in HBM, and reducing the data movement overhead for mode switch between cHBM and mHBM.

Bumblebee consumes 10.9%∼20.1% less memory dynamic energy than state-of-the-art designs on average as Figure 8(d) shows, mainly due to the memory traffic reduction. Besides, Bumblebee consumes less processor energy and memory static (refresh) energy since these energies are mostly proportional to the program runtime.

V. Conclusion

This paper presents Bumblebee, a new hybrid memory system to make HBM serve as either cHBM or mHBM flexibly. The ratio of cHBM to mHBM is adjustable in real time to better exploit both temporal and spatial locality benefits. Bumblebee lowers the over-fetching risk and reduces the data movement cost for mode switch between cHBM and mHBM. In our evaluations, Bumblebee beats the best-previous hybrid memory designs by at least 35.2%, and consumes orders of magnitude less metadata storage space as well as 10.9%∼20.1% less memory dynamic energy.

Acknowledgment

This work is supported by National Key Research and Development Program of China (No. 2022YFB4500303), National Natural Science Foundation of China (NSFC) (No. 62227809), the Fundamental Research Funds for the Central Universities, Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0102), Natural Science Foundation of Shanghai (No. 21ZR1433600), and CCF Huawei Populus Grove Fund (No. CCF-HuaweiSY202202).

References

[1] HBM2. https://www.jedec.org/standards-documents/docs/jesd235a.
[2] DDR4 datasheet. https://www.micron.com/products/dram/ddr4-sdram/part-catalog/mt40a1g8sa-062e.
[3] J. T. Pawlowski. Hybrid memory cube (HMC). In HCS, 2011.
[4] L. Song, et al. Serpens: A High Bandwidth Memory Based Accelerator for General-Purpose Sparse Matrix-Vector Multiplication. In DAC, 2022.
[5] J. B. Kotra, et al. Chameleon: A dynamically reconfigurable heterogeneous memory system. In MICRO, 2018.
[6] J. Sim, et al. Transparent hardware management of stacked DRAM as part of memory. In MICRO, 2014.
[7] J. H. Ryoo, et al. SILC-FM: Subblocked interleaved cache-like flat memory organization. In HPCA, 2017.
[8] A. Prodromou, et al. Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories. In HPCA, 2017.
[9] A. Kokolis, et al. Pageseer: Using page walks to trigger page swaps in hybrid memory systems. In HPCA, 2019.
[10] E. Vasilakis, et al. LLC-guided data migration in hybrid memory systems. In IPDPS, 2019.
[11] P. Behnam, et al. RedCache: Reduced DRAM Caching. In DAC, 2020.
[12] M. K. Qureshi and G. H. Loh. Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design. In MICRO, 2012.
[13] M. N. Bojnordi, et al. ReTagger: An Efficient Controller for DRAM Cache Architectures. In DAC, 2019.
[14] Y. Lee, et al. A Fully Associative, Tagless DRAM Cache. In ISCA, 2015.
[15] D. Jevdjic, et al. Unison cache: A scalable and effective die-stacked DRAM cache. In MICRO, 2014.
[16] X. Yu, et al. Banshee: Bandwidth-efficient DRAM caching via software/hardware cooperation. In MICRO, 2017.
[17] E. Vasilakis, et al. Hybrid2: Combining caching and migration in hybrid memory systems. In HPCA, 2020.
[18] A. Sodani, et al. Knights landing: Second-generation intel xeon phi product. In MICRO, 2016.
[19] T. Sherwood, et al. Automatically characterizing large scale program behavior. In ASPLOS, 2002.
[20] R. Panda, et al. Wait of a decade: Did spec cpu 2017 broaden the performance horizon? In HPCA, 2018.
[21] N. Binkert, et al. The Gem5 Simulator. In ACM SIGARCH, 2011.
[22] P. Rosenfeld, et al. DRAMSim2: A cycle accurate memory system simulator. In IEEE COMPUT ARCHIT L, 2011.
[23] B. Cook, et al. Performance variability on xeon phi. In HPCC, 2017.
[24] S. Singh, et al. Memory Centric Characterization and Analysis of SPEC CPU2017 Suite. In ICPE, 2019.