HBM-Resident Prefetching for Heterogeneous Memory System

Mahzabeen Islam[1], Krishna M. Kavi[1], Mitesh Meswani[⋆2],
Soumik Banerjee[3], and Nuwan Jayasena[3]

[1] University of North Texas, USA
mahzabeenislam@my.unt.edu,krishna.kavi@unt.edu
[2] ARM, USA
mitesh.meswani@gmail.com
[3] Advanced Micro Devices, Inc., USA
{Soumik.Banerjee,nuwan.jayasena}@amd.com

Abstract. To meet the increasing demands for very large memory capacities, bandwidth and energy efficiency, researchers are exploring the use of heterogeneous memory systems that combine faster 3D-DRAMs, DDRx DRAM and non-volatile memories (NVMs). In this paper we evaluate prefetching in a flat-addressable heterogeneous memory comprising High Bandwidth Memory (HBM) and phase change memory (PCM). We find that large prefetch buffers (64MB) can outperform smaller buffer sizes (2MB), however it is not feasible to place such large buffers on the processor die. Hence, in this paper we evaluate an HBM-resident prefetch buffer that provides larger capacity and takes advantage of HBM's higher memory bandwidth. We also present new prefetching policies that accommodate the differences in data path as compared to traditional prefetchers. We show that, reserving a small fraction (1/16th) of HBM memory to host a hardware prefetch buffer can improve IPC for a set of SPEC CPU2006 and HPC benchmarks by an average of 34% and a maximum of 98% over a baseline system with no-prefetching. Prefetching reduces total PCM traffic by 10% on average, which results in more memory traffic to the faster HBM, providing overall performance improvement. We found that such prfetching outperforms CAMEO and Alloy cache schemes on average by 60% and 10%, respectively.

Keywords: Prefetching, heterogeneous memory, HBM, PCM

1 Introduction

Demand for memory performance has been on the rise, especially for data-intensive applications such as HPC, big data analytics, cloud computing and in-memory databases. These applications need memory systems with very large capacities (100s of GBs to TBs), high bandwidth and energy efficiency. For example, SAP HANA in-memory database system requires 256GB to 2TB memory

⋆ The author did the work while employed at AMD.

per host [1]. Conventional DRAM cannot satisfy such capacity and performance demands due power and scaling challenges [2]. Recent 3D-stacked DRAM (3D-DRAM) such as HBM [12] and Hybrid Memory Cube (HMC) [20] provide much higher bandwidth (up to 256GB/s HBM [12] and up to 320GB/s HMC [20]) and consume $\sim$70% [21] less energy than conventional DRAM. There are eight independent channels per HBM stack [12]. However, 3D-DRAM is not likely to meet the capacity requirements of data-intensive applications [3]. Emerging NVM technologies, on the other hand, are much denser, consume low static power and are scalable to provide sufficient capacity [4–8]. PCM is one type of NVM that relies on the state or phase of material to store one bit or multiple bits. Hence it can be much denser than traditional DRAM and provides lower cost-per-bit [22]. Also, PCM consumes less idle state power [13]. PCM may exhibit higher access latency ($\sim$2x for reads and 4x-32x for writes) and higher read (2x) and write energies (4x-140x) than DDRx DRAM [22, 13]. It has limited write endurance of 10^6 to 10^9 cycles [4, 22]. Addressing PCM limitations is an active research area [4–6, 22, 23] and it is believed to be one of the most promising NVMs that can be used as main memory in the near future [2, 4].

As no single memory technology can provide both large capacity and high bandwidth, it is natural to explore heterogeneous memory systems that employ disparate memory technologies together [6, 3, 9–11, 14, 15]. Heterogeneous memory systems introduce their own challenges due to the differences in the characteristics of constituent memory technologies (e.g., storage capacity, access latency, bandwidth, endurance). Recent research has been investigating solutions to these challenges, either employing fast 3D-DRAM memory as a cache for slow memory [14, 15, 10, 6] or employing both fast and slow memories as part of a single physical address space ("flat-address-memory") [3, 9, 11, 6]. Generally, cache-based organizations do not need software changes, but they need to manage large tag space[14, 15]. In cache-based organizations the 3D-DRAM capacity is not included as main memory capacity, which may lead to higher numbers of page faults than flat-address-memory organizations which expose the 3D-DRAM capacity as part of main memory [11]. However, flat-address-memory systems need to employ techniques to efficiently place/migrate frequently accessed data into the faster memory.

Since we are interested in designing large memory systems, in this research, we study a flat-address-memory system consisting in faster 3D-DRAM (HBM) and slower NVM (PCM). We propose to use prefetching methods for bridging the performance gap between 3D-DRAM and NVM. Conventional memory-to-processor prefetching brings data from slower memories (farther from core) to an on-chip buffer (nearer to core). Prefetching can continue to improve performance with larger buffer capacities [16, 10]. For example, in our study of SPEC and HPC workloads (workload details provided in Sect. 4), we find that prefetch buffer sizes of 32MB, 64MB, and 128MB can improve instruction per cycle (IPC) by 27%, 34% and 40% respectively, whereas a 2MB buffer can improve IPC by only 19% over no-prefetching. However, placing such large buffers inside a processor is infeasible due to area and power limitations; processor-chip resident prefetch buffer capacity is typically limited to 1MB to 2MB of SRAM [17]. Hence, in this paper, we propose "HBM-resident" prefetching by setting aside a portion of faster HBM as a large prefetch buffer and employing customized prefetch policies. The prefetch buffer is split over eight HBM channels, allowing high memory level parallelism (MLP). Conventional DRAM provides only a limited number of channels, therefore we choose HBM to host our prefetch buffer. We prefetch data from slower PCM into a buffer space in faster HBM (by storing a copy) and service last level cache (LLC) misses from the buffer on a hit. In case of write-backs from LLC to PCM, if a hit is found in the prefetch buffer, the write is also buffered there. The advantages are that, it provides faster access to data (than accessing it from PCM) while avoiding costly updates to page tables and TLBs (that is generally required in page migration techniques) and reducing write-backs to PCM. The location changes resulting from prefetching is tracked using a hardware-based address remapping table. Another advantage of prefetching is that it might be able to hide PCM access latency even for "cold" misses, as it can predict unseen future addresses. Neither a straightforward demand cache nor a hotness-based page migration scheme can avoid cold misses, as they rely on demand or past history of accesses.

We first evaluate a prefetching scheme that relies entirely on predictability, which is generally known as distance prefetching [18]. Next we present a temporal locality based prefetching technique that relies on access counts to data blocks. We also introduce a simple open-page prefetching policy which can be seen as a relaxed caching policy. The main contributions of this paper are:

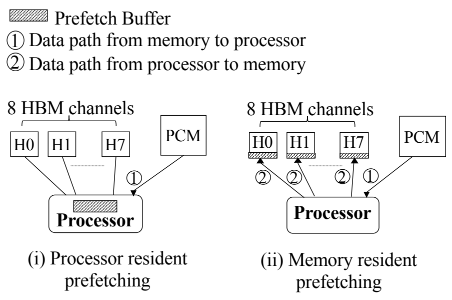
1. Novel "HBM-resident" prefetching hosted in the faster memory to buffer pages of the slower memory.
2. A buffer architecture that is designed to take advantage of memory-level parallelism afforded by the higher number of channels in HBM.
3. Prefetching policies that are designed to take into account the data transfer path and characteristics of emerging memory technologies.

Our studies show IPC improvements of 33% on average (max. 70%) for a set of SPEC CPU2006 workloads and 40% on average (98% max.) for a set of HPC workloads over a baseline system without prefetching. Stand-alone 3D-DRAM-resident prefetching provides an average performance improvement of 60% over a state of the art page migration policy, CAMEO [11], and 10% over Alloy caching [14], which is one of the leading 3D-DRAM based caching techniques.

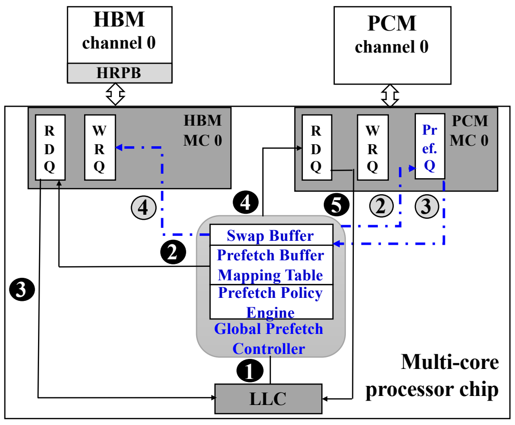
2 Motivation for a New Prefetch Architecture

In conventional prefetching, data from lower level memories (farther from core) are fetched into higher levels (nearer to core) before it is requested by the processor. Some basic hardware prefetching techniques are stride prefetching [24], stream buffers [16], Markov prefetching [25], and Distance prefetching [18].

For emerging memory technologies, different hardware prefetching policies have been explored. Ahn et al. [26] proposed to prefetch data from HMC memory layers into a small SRAM buffer residing in the logic layer of HMC using


▨ Prefetch Buffer
① Data path from memory to processor
② Data path from processor to memory
8 HBM channels
H0
H1
H7
PCM
Processor
(i) Processor resident prefetching
(ii) Memory resident prefetching

(a)



(b)

Fig. 1: (a) Different prefetching organizations, (b) Proposed system organization.

stream prefetching [16] at cache line (64B) granularity. However workloads with good spatial locality can benefit if more cache lines are fetched. In a study by Oskin et al. [10], HBM is employed as an operating system (OS) page cache for conventional DRAM memory, and they employ stride prefetching at OS page (4KB) granularity. Yoon et al. [27] proposed caching PCM row buffers with high access and conflict counts into conventional DRAM to avoid repeated opening of the same row in PCM. However, memories with high MLP distribute their cache blocks from the same physical page to a number of channels and, as such, tracking row buffer conflicts may no longer be beneficial since the locality is now spread over channels and they may not conflict. In our evaluation we configure each type of memory with cache-line-level address interleaving to make most of the MLP. Previously we proposed to use a customized distance prefetching policy to prefetch from slower PCM to a processor-chip-resident small SRAM buffer (2MB) [19]. Here, unlike the processor-resident prefetch buffer, we propose to host a much larger (32x) prefetch buffer in HBM using only 1/16th of the total HBM capacity. This approach not only eliminates the capacity constraints, but also allows high bandwidth utilization through MLP since the prefetch buffer is split over multiple channels of HBM.

3 HBM-Resident Prefetching

3.1 Architecting a HBM-Resident Buffer

Fig.1(a) provides a comparison of high level organizations for processor-resident and HBM-resident prefetch buffers for a flat-address memory system comprising the HBM and PCM used in our evaluations. While designing our prefetch architecture we addressed four design parameters: **i) prefetch buffer location, ii) prefetch granularity, iii) prefetch initiation and iv) prefetch policies**.

Prefetch buffer location dictates where to host the prefetch buffer and its associated data path for copying data to the buffer, which directly influences the cost/time for completing the prefetch operation. In our design, we host the prefetch buffer in the HBM and call it HBM-resident prefetch buffer (HRPB). We compare our design with processor-chip-resident SRAM prefetch buffer, referred to as on-chip prefetch buffer (OCPB), as proposed in [19]. HRPB necessitates a different data path for bringing data to the buffer as shown in Fig.1(a). For OCPB, data travels one-way from memory to the processor buffer but, for HRPB, data travels first from slower memory to a temporary swap buffer in processor (not shown in the figure) and from there to the memory buffer. This is because there is currently no direct data path from PCM to HBM. **Prefetch granularity** influences the cost for storing and accessing tags for prefetched data. Since the HRPB is large, the tag array size can grow very large when prefetching at finer granularity. Therefore, we use a coarser block (2KB) granularity. **Prefetch initiation** dictates when to issue a prefetch request; we take an "opportunistic" approach and only prefetch (read) from the PCM when it is not busy serving demand read requests. Finally, we have to design **prefetch policies** to amortize the cost of the longer data path and the difference in buffer storage technology. Details on the polices are described in Sect. 3.3.

3.2 System Organization

Fig. 1(b) shows the system organization of our prefetching technique. The multi-core processor chip has a shared LLC and a set of memory controllers (MCs). We use 8 HBM and 2 PCM channels with one MC per channel. Fig. 1(b) shows details of one HBM MC and one PCM MC to keep the figure readable. Each MC contains a read queue (RD Q) and write queue (WR Q), and the PCM MC also contains a Prefetch queue (Pref. Q). We reserve a small fraction of the HBM as HRPB (e.g., 64MB of 1GB HBM), which is not visible to the OS and hence non-allocable. The hardware-based global prefetch controller is located on the processor chip. The HRPB address range is only visible to the prefetch controller. We have assumed that the OS-visible physical address range is statically partitioned over HBM (excluding the HRPB portion) and PCM.

We prefetch (copy) at 2KB block granularity from the PCM to the HRPB. The original block is still kept in PCM so no page remapping is required. We store the HRPB tag array inside the global prefetch controller. The tag array also serves as the prefetch buffer mapping table. The HRPB is a 4-way set associative, write-back buffer with least recently used (LRU) eviction policy. In the mapping table, with each 37 bit address tag (which is the original PCM physical address of that 2KB block) we store 1 valid bit, 1 dirty bit and a 32 bit vector for tracking dirty cache lines in the prefetched block. Hence, for a 64MB HRPB, the mapping table size will be 288KB (32,768 entries, each 9B), which is feasible to place on the processor chip. Since each entry in the mapping table corresponds to a fixed physical address in the HRPB, on a hit in the mapping table, the HRPB physical address can be dynamically generated. While evicting a block from the HRPB, only the dirty cache lines are written back to the PCM. The prefetch policy engine implements the policies.

How to access prefetched data in HRPB: On every LLC miss, the address is redirected to the prefetch controller. The controller first checks the missed address to see if it is a PCM address (we assumed that the physical address range is statically partitioned between HBM and PCM) and, if it is, then the controller looks up the mapping table. If a match found, then the new destination address inside the HRPB is also known, and the request is directed to HRPB. The solid path with numbers 1, 2, and 3 in Fig. 1(b) shows this path. Unlike traditional migration techniques, the missing data is not moved to the OS-visible memory space of the HBM (hence avoiding costly page table remappings), but kept in the prefetch buffer. If no match is found in the mapping table, the LLC miss request is serviced by the PCM (in Fig. 1(b) solid path 4 and 5). The delay overhead for every miss in the mapping table is fairly small: the time to access a 288KB on-chip SRAM array for a 64MB HRPB.

How to prefetch data from PCM to HRPB: For every LLC miss to PCM, the policy engine will generate the next address to prefetch depending on the prefetching policy. The prefetch controller first checks if the generated address is already present in the HRPB. Otherwise, it generates prefetch read requests to the PCM and reads that block into a swap buffer located inside the prefetch controller. The swap buffer holds one block of prefetched data (2KB). The prefetch controller then finds a destination location in the HRPB by checking the mapping table (if needed, write-back of the evicted entry takes place first). After successful writes to HRPB, the mapping table entry is updated with the new block's address tag and the valid bit is set. This flow is shown in Fig. 1(b) by dotted paths numbered 2, 3, and 4.

3.3 Prefetching Policies

Distance prefetching is a generalization of Markov prefetching that relies on correlating deltas (differences) between addresses [18]. Similar to [19], we choose to use the global history buffer (GHB) structure to implement the distance prefetcher as presented by Nesbit et al. [28]. Storage overhead for implementing GHB for distance prefetching is only 8KB [28]. Distance prefetching width degree determines how many different prefetching paths and depth degree determines how far into the future we want to explore. We found width degree 1 and depth degree 4 as optimal for us.

In **Hotness-based prefetching**, we use the "hotness" metric (a count of the number of accesses to a block) [3, 6] in deciding whether to prefetch a block into HRPB. Whenever a block is accessed more than a certain number of times (e.g., 4) we immediately generate a prefetch request for that block. Such temporal locality based prefetching may provide higher confidence that the prefetched data will be useful. We use a hotness count cache with only 16K (16,384) entries to hold the hotness count of recently accessed blocks, it works in similar manner as filter cache presented in the CHOP study [29]. Each entry of the hotness count cache is 6B (37b address tag and the rest to keep hotness count), hence the size of the hotness count cache is only 96KB, which can be stored on the processor chip and accessed with small delay overhead in the prefetching path.

In **Open-page prefetching**, to benefit from row buffer locality, we change our physical memory address interleaving from cache line level to memory page (e.g., row with size 2KB) level granularity so that each 2KB sized block falls to the same row. To minimize opening the same row in PCM repeatedly, we employ a simple prefetching policy that attempts to prefetch any row buffer that is open. This can be seen as a relaxed caching policy, exploiting spatial locality.

4 Experimental Setup

We assume a 16-core system with main memory comprising 1GB HBM and 16GB PCM in a flat-address model. Each of the cores is 4-wide out-of-order issue with 128 entry ROB and operates at 3.2GHz. Each core has private L1 I (32KB) and D (16KB) caches, and all 16 cores share an L2 LLC (16MB). For HBM and PCM timing parameters we primarily follow [30] and [31] respectively; we list them in Table 1. As a baseline, we used above mentioned memory system without any prefetching or data migration.

We use Ramulator [30] in trace-driven mode with a CPU model to estimate IPC. To generate the traces, we first use PinPlay kit [32] to identify region of interest (ROI) of one billion instructions for each of the benchmarks in Table 2. We have used 17 memory-intensive benchmarks from the SPEC CPU2006 suite [34], and four representative HPC benchmarks from the US Department of Energy: XSBench [35], LULESH [36], CoMD [37] and miniFE [38]. As listed in Table 2, we generate memory access traces for twenty multi-programmed workloads by running 16 copies of ROI traces of one benchmark or ROI traces from different random benchmarks in a 16-core Moola cache simulator [33].

5 Evaluation

5.1 Performance Analysis

We first present the IPC performance improvements of the three prefetching policies, namely Distance (delta), Hotness-based (hot) and Open-page (open) implemented with HRPB as well as delta with OCPB [19]. The OCPB can be seen as an LLC prefetcher which uses a separate on-chip buffer area for prefetching to avoid the risk of polluting the LLC. Details of the prefetching configurations are provided in tables 3 and 4. For the figures 2, 3 and 4, the positive y-axis shows IPC percentage improvement whereas the negative y-axis shows IPC degradation with respect to a baseline system without any prefetching or migration. We categorize the workloads 1 to 10 as listed in Table 2 as SPEC homogeneous (SPEC_HOM), 11 to 14 as HPC homogeneous (HPC_HOM), and 15 to 20 as SPEC heterogeneous (SPEC_HET).

Fig. 2 shows the IPC improvements for different prefetching policies over the baseline. For our basic set of experiments we have chosen the HRPB size as 64MB and hot policy threshold as 4 after performing capacity and threshold sensitivity analyses (not presented here due to space limitations). The HR_hot

Table 1: Baseline configuration

Parameter	**HBM**	**PCM**
Channels, capacity	8, 1 GB (8 x 128 MB)	2, 16 GB (2 x 8 GB)
Memory Controller (MC)	1 per channel	1 per channel
Row buffer size	2 KB	2 KB
Queue size/MC	RD 32, WR 32 entries	RD 64, WR 256, and prefetch 32 entries
Latency	tCAS-tRCD-tRP-tRAS 14ns-14ns-14ns-34ns	Read 80ns (7.5ns tPRE+62.5ns tSENSE+10ns tBUS) Write 250ns tCWL
Bus (per channel)	128-bit, 500MHz	64-bit, 400MHz

Table 2: Evaluated workloads (WL); footprint (FP) is provided in GB

No.	**WL**	**Benchmarks**	**MPKI**	**FP**	**No.**	**WL**	**Benchmarks**	**MPKI**	**FP**
1	mcf	16x mcf	65.04	16	8	bwav	16x bwaves	6.90	6.82
2	lbm	16x lbm	44.21	6.30	9	cactus	16x cactusADM	3.70	2.31
3	milc	16x milc	23.05	9.05	10	xbmk	16x xalancbmk	4.50	2.89
4	omntp	16x omnetpp	18.96	2.06	11	xsb	16x XSBench	22.01	14.68
5	astar	16x astar	16.80	2.63	12	lul	16x LULESH	13.51	6.80
6	gems	16x GemsFDTD	9.59	10.59	13	mini	16x miniFE	6.72	10.66
7	zmp	16x zeusmp	8.14	3.32	14	comd	16x CoMD	1.41	2.30

No.	**WL**	**Benchmarks**	**MPKI**	**FP**
15	mix1	3x mcf, sph., 2x ast., 2x lbm, gcc, 2x sop., lib., 2x milc, omn., libq.	29.36	5.64
16	mix2	3x lbm, 2x mcf, 3x deal., 3x sop., bzi., 2x cac., 2x Gem.	20.47	5.08
17	mix3	2x Gem., lib., 2x milc, deal., 2x sph., 2x les., 2x cac., 2x gcc, bzi., ast.	10.99	3.34
18	mix4	mcf, 3x lib., 3x sop., Gems., milc, les., lbm, gcc, 2x bzi., cac., deal.	18.27	3.60
19	mix5	5x mcf, 6x lbm, 5x sop.	42.51	7.61
20	mix6	4x lib., 3x omn., 3x gcc, 3x sph., 2x milc, ast.	19.64	2.11

Table 3: Buffer Configuration

Legends	**Details**
HRPB	64MB, 4-way, write-back, LRU eviction
OCPB	2MB, 16-way, write-back, LRU eviction

Table 4: Policy Configuration

Legends	**Details**
OC_delta	OCPB with Distance policy, width=1, depth=4
HR_delta	HRPB with Distance policy, width=1, depth=4
HR_hot	HRPB with Hotness-based policy, threshold 4
HR_open	HRPB with Open-page policy

scheme provides the best average result over all other policies for all three categories of workloads SPEC_HOM, SPEC_HET and HPC_HOM with average IPC improvements of 27%, 43% and 40% respectively. HR_hot generally works well as usually a block with frequent accesses is a good predictor of that same block being accessed in the future. We find that HR_hot performs the best for 1/2 of the workloads. For most of these cases HR_hot policy's prefetch accuracy and repeated hit to the same block is much higher than the other policies.

The prediction-based delta policy works well when the workloads have predictable sequences which happens when there are repeating sequence of address strides. OC_delta [19] follows a similar trend as HR_delta but since OCPB is much smaller in size (only 1/32th of HRPB), we observe on average smaller improvement for OC_delta. For 1/3rd of the workloads, OC_delta provides per-

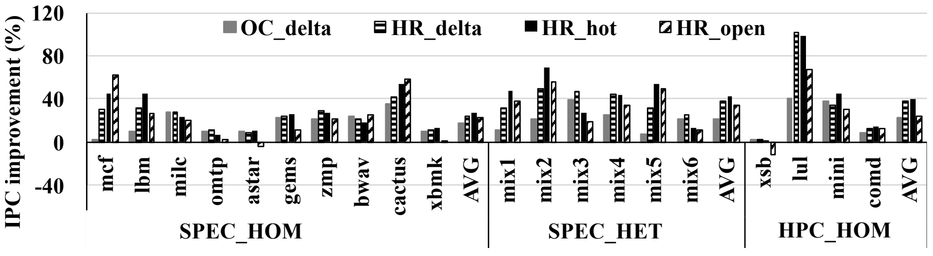


Fig. 2: IPC improvement (%) of different prefetching policies over baseline (negative y-axis shows degradation)

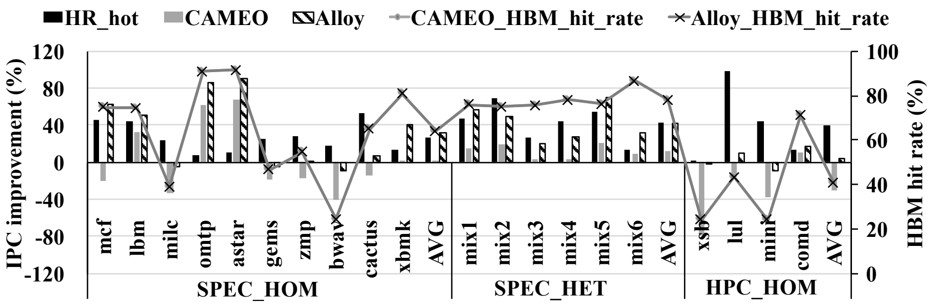


Fig. 3: IPC improvement (%) of HR_hot prefetching, stand-alone CAMEO, and stand-alone Alloy cache over baseline system

formance close to or better than HR_delta. We found that these workloads have diminishing reuse of data with time, hence storing many prefetched blocks for prolonged periods of time does not help improve the performance.

In case of simpler HR_open policy, we have changed the memory address interleaving from cache-line-level to block-level to achieve more row buffer hits. This decreases the MLP and hence we see smaller performance improvement for most of the cases. However, for mcf and cactus our analysis shows that the majority of the blocks have low reuse distance and the HR_open policy is the quickest to initiate a prefetch request since it simply tries to prefetch the most recently accessed row buffer and hence it outperforms the HR_hot policy.

With HBM-resident prefetching, on average, total PCM traffic is decreased by 10%, compared to the no-prefetching baseline. Fewer accesses to PCM leads to better average access latencies and reduced energy consumption.

Comparison with CAMEO and Alloy: Here, we compare our best prefetching policy, HR_hot, with CAMEO page migration technique [11] and Alloy caching [14]. Chou et al. proposed CAMEO (CAche-like MEmory Organization) for a two level memory system comprising 3D-DRAM and DDR DRAM [11]. The 3D-DRAM stores recently accessed data by employing a “cache-like” migration policy, but it is visible to the OS. On a 3D-DRAM demand miss, the requested line (64B) is filled from DDR DRAM. To make room for the requested line, an older line needs be written back to DDR DRAM (even if it is not dirty) since there is no other copy of this line in memory. We use CAMEO model with HBM and PCM, with a capacity ratio of 1:16. In Alloy cache, faster 3D-DRAM

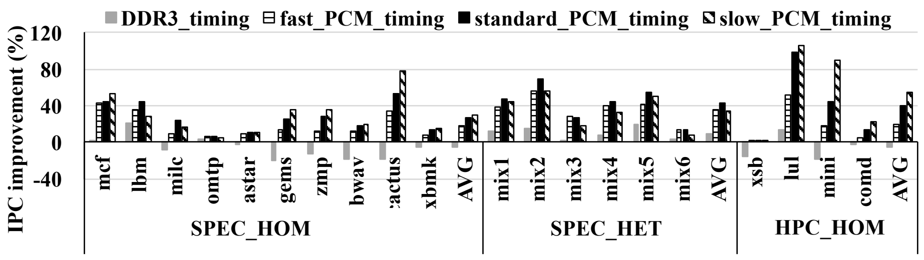


Fig. 4: PCM timing sensitivity for Hotness-based prefetching policy

is employed as a large LLC to slower conventional DRAM memory [14]. The 3D-DRAM is employed as a direct-mapped cache with 64B line granularity. Here both tag and data are kept together. In our implementation we use HBM as Alloy cache to slower PCM memory. In CAMEO, we have total 17GB of main memory, whereas in Alloy chaching we have 16GB of main memory. Since each of our workload's memory footprint is below 16 GB, we cannot see the larger capacity benefit of CAMEO over Alloy caching.

In Fig. 3, right y-axis shows the HBM hit rate (%) and the hit_rate lines correspond to it. On average CAMEO degrades IPC by 1%, Alloy cache improves IPC by 29% and HR_hot improves by 34%. In case of CAMEO, every HBM miss results in a write back to PCM and, as a result, HBM hit rate plays an important role in CAMEO's performance. Generally for workloads with HBM hit rate under 74% we see performance degradation with CAMEO. Though writes are not in the critical path of execution, when the write queues are almost full, memory controllers must prioritize write queues over read queues and hence the overall execution time can be slowed down. Here, we have used different memory technology and capacity ratios than proposed in the original CAMEO work [11], and thus due to the high memory pressure in 3D-DRAM and slow writes of PCM, we observe very little performance improvement by CAMEO. In case of Alloy cache, only dirty cache lines are written back to PCM, hence Alloy cache with similar HBM hit rate provides better performance than CAMEO.

5.2 PCM Timing Analysis

Fig. 4 shows how sensitive our proposed HBM-resident Hotness-based prefetching is to the PCM timing (due to space limitation we do not include the results for other two prefetching policies; in general they follow a similar trend). In one extreme we have replaced PCM with conventional DRAM (DDR3). Also, we evaluate a 2x faster fast_PCM and a 2x slower slow_PCM taking the PCM timing mentioned in Table 1 as standard. In Fig. 4 the IPC improvements for each timing configuration are compared to the baselines with identical timing.

When we prefetch from DDR DRAM to HBM buffers, we do not see significant benefits because both memories have similar access latencies. However, HBM has more channels and can provide more MLP than conventional DRAM. Hence, in this case we are paying the prefetching cost only to get the higher bandwidth benefit of HBM. From Fig. 4 we can see that for ∼2/3rd of the workloads we achieve negligible IPC improvements or degradations and for the rest we achieve IPC improvements from 8% to 21%. With fast PCM, the amount of time we save on a hit in the HRPB is smaller than the case when we have standard PCM. Hence with fast PCM, we achieve smaller performance improvements. On the other hand, with the slow PCM, we have fewer opportunities to prefetch since PCM is mostly busy responding to demand requests.

6 Conclusion and Future Work

We presented a novel HBM-resident hardware-based prefetching mechanism for heterogeneous flat-address-memory comprising HBM and PCM. We evaluated three different prefetching policies and show that they perform better than a system with no prefetching. In the future, we will explore composite schemes by augmenting such prefetching policies with data migration and caching organizations for heterogeneous memories. Further, HBM-resident prefetch buffer can be employed as a staging area to make the final migration decision of the page to the faster memory. Hence we believe this research opens new opportunities involving prefetching in the context of heterogeneous memory.

References

1. HANA Memory Usage, `http://saphanatutorial.com/sap-hana-memory-usage-explained/`
2. Mutlu, O.: Memory scaling: A systems architecture perspective. In: International Memory Workshop. IEEE (2013)
3. Meswani, M.R., et al.: Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In: HPCA, pp. 126–136. IEEE (2015)
4. Qureshi, M.K., et al.: Phase Change Memory: From devices to Systems. Synthesis Lectures on Computer Architecture, vol. 6, no. 4, pp. 1–134. (2011)
5. Qureshi, M.K., et al.: Scalable high performance main memory system using phase-change memory technology. In: ACM SIGARCH Computer Architecture News, vol. 37, no. 3, pp. 24–33. (2009)
6. Su, C., et al.: Hpmc: An energy-aware management system of multi-level memory architectures. In: MEMSYS, pp. 167–178. ACM (2015)
7. Micron NVDIMM, `https://www.micron.com/products/dram-modules/nvdimm#/`
8. 3D-XPoint, `http://www.intel.com/newsroom/kits/nvm/3dxpoint/pdfs/Launch_Keynote.pdf`
9. Sim, J., et al.: Transparent hardware management of stacked dram as part of memory. In: MICRO, pp. 13–24. IEEE (2014)
10. Oskin, M., Loh, G.H.: A Software-managed Approach to Die-stacked DRAM. In: PACT, pp. 188-200. IEEE (2015)
11. Chou, C., et al.: CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In: MICRO, pp. 1–12. IEEE Computer Society, (2014)
12. 3D-ICs, `https://www.jedec.org/category/technology-focus-area/3d-ics`
13. Numonyx: PCM, `http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf`

14. Qureshi, M.K., Loh, G.H.: Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design. In: MICRO, pp. 235–246. IEEE Computer Society (2012)
15. Jevdjic, D., et al.: Unison cache: A scalable and effective die-stacked dram cache. In: MICRO, pp. 25–37. IEEE (2014)
16. N. P. Jouppi: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: ISCA, pp. 364–373. IEEE (1990)
17. Beckmann, N., Sanchez, D.: Meeting midway: improving cmp performance with memory-side prefetching. In: PACT, pp. 289–298. IEEE (2013)
18. Kandiraju, G.B., Sivasubramaniam, A.:Going the distance for TLB prefetching: an application-driven study. In: IEEE Computer Society, vol. 30. (2002)
19. Islam, M., et al.: Prefetching as a Potentially Effective Technique for Hybrid Memory Optimization. In: MEMSYS. ACM (2016)
20. Hybrid Memory Cube Consortium, `http://www.hybridmemorycube.org/`
21. Kim, J., Kim, Y.: HBM: Memory Solution for Bandwidth-Hungry Processors. In: Hot Chips: A Symposium on High Performance Chips. (2014)
22. Yoon, H., et al.: Efficient data mapping and buffering techniques for multilevel cell phase-change memories. In: TACO, vol. 11, no. 4, p. 40. ACM (2015)
23. Wang, H., et al.: Duang: Fast and lightweight page migration in asymmetric memory systems. In: HPCA, pp. 481–493. IEEE, (2016)
24. Fu, J.W., et al.: Stride directed prefetching in scalar processors. In: ACM SIGMICRO Newsletter, vol. 23, no. 1-2, pp. 102–110. (1992)
25. Joseph, D., Grunwald, D.: Prefetching using markov predictors. In: ACM SIGARCH Computer Architecture News, vol. 25, pp. 252–263. ACM (1997)
26. Ahn, J., et al.: Low-power hybrid memory cubes with link power management and two-level prefetching. In: Transactions on VLSI Systems, 24(2), 453-464. IEEE (2016)
27. Yoon, H., et al.: Row buffer locality aware caching policies for hybrid memories. In: International Conference on Computer Design, pp. 337–344. IEEE (2012)
28. Nesbit, K.J., Smith, J.E.: Data cache prefetching using a global history buffer. In: IEE Proceedings Software, pp. 96–96. IEEE (2004)
29. Jiang, X., et al.: Chop: Adaptive filter-based dram caching for cmp server platforms. In: HPCA, pp. 1–12. IEEE (2010)
30. Kim, Y., et al.: Ramulator: A fast and extensible dram simulator. In: Computer Architecture Letters. (2015)
31. Nair, P.J., et al.: Reducing read latency of phase change memory via early read and turbo read. In: HPCA, pp. 309–319. IEEE (2015)
32. Intel PinPlay, `https://software.intel.com/en-us/articles/program-recordreplay-toolkit`
33. Shelor, C.F., Kavi, K.M.: Moola: Multicore Cache Simulator. In: International Conference on Computers and Their Applications. (2015)
34. SPEC CPU 2006, `https://www.spec.org/cpu2006/`
35. Proxy-Apps for Neutronics, `https://cesar.mcs.anl.gov/content/software/neutronics`
36. Lawrence Livermore National Laboratory: Hydrodynamics Challenge Problem. In: Tech. Rep. LLNL-TR-490254.
37. Mohd-Yusof, J., et al.: Co-design for molecular dynamics: An exascale proxy application. (2013)
38. Heroux, M., Hammond, S.: MiniFE: Finite Element solver, `https://portal.nersc.gov/project/CAL/designforward.htm#MiniFE`