Prefetching as a Potentially Effective Technique for Hybrid Memory Optimization

Mahzabeen Islam
University of North Texas
Denton, TX, USA
MahzabeenIslam@my.unt.edu

Soumik Banerjee
AMD Research
Advanced Micro Devices, Inc.
Austin, TX, USA
Soumik.Banerjee@amd.com

Mitesh Meswani[*]
AMD Research
Advanced Micro Devices, Inc.
Austin, TX, USA
mitesh.meswani@gmail.com

Krishna Kavi
University of North Texas
Denton, TX, USA
Krishna.Kavi@unt.edu

ABSTRACT

The promise of 3D-stacked memory solving the memory wall has led to many emerging architectures that integrate 3D-stacked memory into processor memory in a variety of ways including systems that utilize different memory technologies, with different performance and power characteristics, to comprise the system memory. It then becomes necessary to manage these memories such that we get the performance of the fastest memory while having the capacity of the slower but larger memories. Some research in industry and academia proposed using 3D-stacked DRAM as a hardware managed cache. More recently, particularly pushed by the demands for ever larger capacities, researchers are exploring the use of multiple memory technologies as a single main memory. The main challenge for such flat-address-space memories is the placement and migration of memory pages to increase the number of requests serviced from faster memory, as well as managing overhead due to page migrations.

In this paper we ask a different question: can traditional prefetching be a viable solution for effective management of hybrid memories? We conjecture that by tuning well-known prefetch mechanism for hybrid memories we can achieve substantial performance improvement. To test our conjecture, we compared the state of the art CAMEO migration policy with a Markov-like prefetcher for a hybrid memory consisting of HBM (3D-stacked DRAM) and Phase Change Memory (PCM) using a set of SPEC CPU2006 and several HPC benchmarks. We find that CAMEO provides better performance improvement than prefetching for $2/3^{\text{rd}}$ of the workloads (by 59%) and prefetching is better than CAMEO for the remaining $1/3^{\text{rd}}$ (by 19%). The EDP analysis shows that the prefetching solution improves EDP over the no-prefetching baseline whereas CAMEO does worse in terms of average EDP. These results indicate that prefetching should be reconsidered as a supplementary technique to data migration.

[*]This author did this work while working in AMD Research.

MEMSYS '16, October 03-06, 2016, Alexandria, VA, USA

© 2016 ACM. ISBN 978-1-4503-4305-3/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2989081.2989129

CCS Concepts

•**Computer systems organization → Heterogeneous (hybrid) systems;**

Keywords

Prefetching; hybrid memory system

1. INTRODUCTION

The demand for memory capacity and bandwidth continues to rise faster than what current DRAM (DDR) memory systems can provide. However, there have been some new memory technologies that may address these needs. These include 3D-stacked DRAM (3D-DRAM) for providing performance and non-volatile memories (NVM), including Phase Change Memories (PCM) for providing high capacity at a low cost [29, 31, 36]. Vendors have already announced systems with either integrated 3D-DRAM or as off-chip memories [20, 8, 26] and in the coming years NVM is likely to replace or augment DRAM in a system. Emerging non-volatile DIMM (NVDIMM) standards [22] and recent industry announcements of 3D Xpoint [9] and Phase Change Memory (PCM) [32, 5] provide larger capacity than conventional DRAM and better performance than flash based NVM but are still much slower than conventional DRAMs. Also higher write energy and lower endurance are still challenges for PCM. As a result we expect that in coming years systems will have a variety of memory technologies to address the performance, power and cost challenges resulting in a heterogeneous (hybrid) memories [36, 21, 34, 27, 4, 30, 11].

The ultimate goal, as traditional caches, is to make the average memory access equal to that of the faster main memory while providing the larger capacity of the slower memory. There has been significant research in this area [36, 21, 34, 27, 4] which primarily focuses on either managing fast memory as a cache for slower memory or using both fast and slow memories as part of a single address space ("flat address space") with direct load/store access. In cache-based configurations [30, 11], research deals with storing and efficiently accessing large number of tags for GBs of memory. An obvious advantage of cache-based schemes is that existing software can run without any changes as 3D-DRAM is just another level of cache. However, in this case the memory capacity of faster 3D-DRAMs is not visible to the operating system (OS) and is not available for applications to allocate space there. Flat-address-space configurations [36, 21, 34, 4] have focused on policies for migrating (moving) frequently accessed "hot" pages from a slower memory to the faster memory and efficient methods to update the page mappings (required for correct address translation). While efficient migration and remapping methods are challenging, exposing all memory capacity will benefit capacity-constrained applications [21, 4]. The challenge with flat-address-space schemes lies in maintaining auxiliary structures to support profiling individual pages (for 100 GB memory with 4K pages and 8-bit counters results in approximately 20 MB storage), and intelligent mechanisms to reduce the impact of costly page remapping (page table updates and translation lookaside buffer-TLB shootdowns). Moreover, in order to adapt quickly to changes in access behaviors of applications, these schemes need to profile access at a finer granularity than page granularity.

In this research, we posit that by revising conventional prefetching techniques, we can provide an additional and potentially less complex solution to bridge the performance gap between faster and slower memories. NVM can provide higher capacity than conventional DRAM but comes with other performance constraints, hence in our study we want to hide the drawbacks of NVM by exploiting advantageous features of 3D-DRAM. In this paper we study a hybrid flat address space memory consisting of HBM and PCM. We first present hardware based Distance prefetching [14] which is a Markov-like prefetching policy [12] specially adapted for hybrid memory systems and then compare the prefetching solution with the state of the art CAMEO model [4] which employs both hardware caching and migration techniques for such hybrid memories. Our experiments show that on average prefetching and CAMEO improve performance by 15% and 50% respectively over no-prefetching baseline. CAMEO outperforms prefetching for $2/3^{rd}$ of the workloads by 59% whereas for the remaining $1/3^{rd}$ workloads, prefetching outperforms CAMEO by 19%. Moreover, as compared to the no-prefetching baseline Energy Delay Product (EDP), prefetching EDP is improved by 8% and CAMEO EDP is degraded by 2.5 times on average. Hence, we feel that prefetching can be a promising technique that needs to be revisited to optimize hybrid memory systems.

2. BACKGROUND

2.1 Emerging Memory Technology

3D-Stacked DRAM is a new memory technology where multiple DRAM dies are stacked vertically and are connected by high density through-silicon vias (TSVs). As TSVs are essentially on-chip connections, they can provide very high bandwidth, consuming much less energy compared to off-chip memory accesses. Each memory stack also provides a greater number of channels (8 in the case of HBM) compared to a single channel provided by a conventional DDR-DRAM memory module. It has been reported that a 3D-DRAM provides 8x higher bandwidth and consumes $\sim$70% less energy than conventional DRAM [16]. Thus 3D-DRAM is a suitable memory technology to satisfy memory bandwidth requirements of data-intensive applications.

Phase Change Memory (PCM) is one of the most promising and widely-studied NVMs [29, 31, 36, 15]. It can be much denser than conventional DRAMs, particularly when multiple bits can be stored in a single cell (Multi-Level Cell, or MLC PCM) and consumes low static power. However, it has higher access latency ($\sim$2x for reads and 5x-32x for writes), consumes higher cell access energy (2x for read and 10x-140x for writes) when compared to conventional DRAM, depending on the number of bits/cell [39]. PCM has limited write endurance [29]. However, PCM is a promising memory technology since it can satisfy large memory capacity requirements of data-intensive applications.

2.2 Background on Heterogeneous Memory Systems

There have been a number of research efforts on managing and architecting heterogeneous memory systems comprised of two or more different memory technologies with different characteristics. In some studies, the faster memory (either 3D-DRAM with respect to traditional DRAM or DRAM with respect to PCM) has been used as a cache for the slower memory [36, 21, 34, 27, 4, 30, 11]. One of the immediate advantages of a cache approach is that existing software can run unmodified, as 3D-DRAM (HBM) is just another level of cache. One of the challenges of managing a multi-gigabyte cache is efficiently accessing and storing the large number of tags (e.g., for a 4GB direct-mapped HBM cache with 64B line size, tag storage is approximately 384MB) that cannot fit in an on-chip SRAM. Consequently, there has been a large body of research that proposed methods to maintain the large tag space and to efficiently perform tag accesses [30, 15]. One of the major drawbacks of using HBM as a cache is that the HBM capacity (several GBs) is not visible to software, which can be detrimental for memory-capacity-constrained applications [4]. Moreover, when HBM is used as a cache, the available memory bandwidth (from processor's point of view) is limited by the HBM bandwidth, while exposing both memory types can yield improved performance due to the combined bandwidth and memory-level parallelism (MLP).

There are other studies that manage two different memory technologies as part of a single physical address space that is visible to the OS. For such system organizations the main challenge is to intelligently place and migrate data between the different memories to ensure optimal performance and energy efficiency. Meswani *et al.*, [21] proposed minimal hardware that tracks page access counts to identify most frequently accessed "hot" pages, while others use demand driven [4] approaches or set dueling methods [34] to trigger page migrations. When we migrate a page into faster memory, an infrequently accessed "cold" page is evicted and migrated from faster to slower memory (unlike a cache, which stores a copy of the page). Since PCM has limited write endurance, higher write latency and higher write energy, evictions to PCM will add overheads, and decrease the life of PCM memories. In such systems it is also necessary to address the challenge of page remapping to maintain correct physical addresses across page migrations by updating TLBs and page tables: these activities can be very costly (10s of microseconds [21, 27]). Some proposed to lower the remapping cost by migrating at very coarse intervals of 100s of milliseconds [21], or using cache-like methods [4] or remapping tables in the memory controller (MC) [34].

In our research, we propose to prefetch pages from slower memories into an on-chip (core-side) small prefetch buffer, instead of migrating pages between faster and slower memories. The advantage of prefetching over migration is that it provides faster access while avoiding costly updates to page tables and TLBs. It also avoids writing any clean copy of data from core-side prefetch buffer back to PCM. While traditional caching also makes a copy and has similar advantages as our buffered data, the difference lies in when and which data is prefetched into the buffer. Caches fetch on a demand miss and the data that is cached may or may not exhibit temporal locality. Our prefetching policies rely on analyzing memory access patterns and predicting data usage and prefetch it in advance of a demand for it. Unlike caching, prefetching is done opportunistically only when the memory is not busy serving the demand misses. Thus, the policy for data fetching is the key difference between caching and prefetching.

2.3 Hardware Prefetching Overview

2.3.1 *Traditional Hardware Prefetching Background*

Traditional on-chip prefetching mechanisms attempt to pro-actively fetch data from lower level memories (farther from core) to higher level memories (nearer to core) before it is requested by the processor. Stride prefetching [6], stream buffers [13] and Markov prefetching [12] are some common approaches for cache-level hardware prefetching. Stride prefetching relies on a hardware structure that detects recurring strides in the sequence of load addresses. This structure is then typically indexed by the program counter (PC) of the current instruction to predict the stride of future accesses. Depending on the prefetching degree N, the prefetcher then brings the next N elements based on the predicted stride. Markov prefetching relies on address correlation prefetching and stores the history of the miss address stream. For example, we have below miss address stream-

miss address stream: a, d, x, m, a, p, j, j, a

There is one entry for each unique miss address (e.g., “a”) in the correlation table and the entry also records one or more miss addresses (e.g., “d” and “p”) that followed the first miss address (“a”). When address “a” misses again, the table is indexed using this address to find out which miss addresses followed “a” last time (in our example they are “d” and “p”). Distance prefetching [14], another kind of correlation prefetching can be seen as generalized Markov prefetching. The correlation table stores one entry for each unique “delta” between any two consecutive miss addresses instead of storing the miss address itself. Each entry records the history of deltas seen right after the current “delta” instead of recording the actual miss addresses as done in Markov prefetching. The correlation table is indexed by deltas of the global miss addresses.

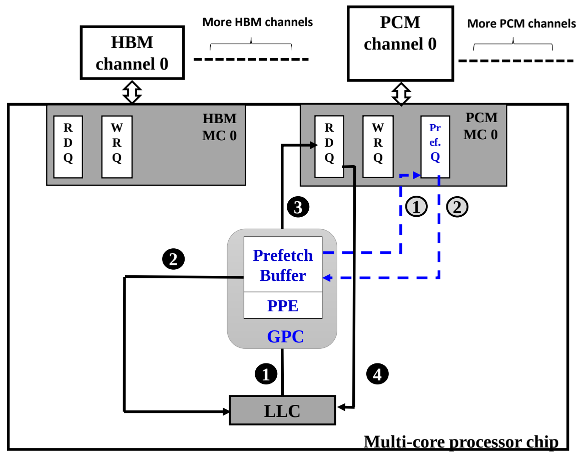

Figure 1: High level organization of on-chip prefetcher (figure is not drawn to the scale).

2.3.2 *Prefetching for Hybrid Memories*

A number of publications have explored prefetching for emerging memory technologies. Ahn *et al.* [3] proposed a two-level prefetcher for systems with Hybrid Memory Cube (HMC) as main memory. One of them is a conventional on-chip prefetcher between last-level cache (LLC) and memory. The other is a specialized prefetcher that prefetches from HMC DRAM memory layers to a small SRAM prefetch buffer within the HMC module. In this design, in-memory prefetching employs a simple stream prefetcher [13] at cache line (64 byte) granularity with a prefetch degree 4. Oskin *et al.* [27] have proposed the use of stride prefetching while using HBM as an OS page cache for conventional DRAM memory. Stride prefetching at page size (4KB) granularity with small prefetch degree was found to be beneficial. Yoon *et al.* proposed [38] to use conventional DRAM as a row buffer cache for PCM. This work caches the PCM row buffers that have high access counts but also incur more row conflicts (i.e., have lower temporal locality) in DRAM to save repetitive activation of the same row.

3. METHODOLOGY

In this section, we first discuss the organization of the on-chip (core-side) prefetcher and then provide a brief overview on the prefetch technique and the data migration technique that we have studied. We evaluate a baseline system with HBM and PCM memory as part of the same flat-address-space; details are provided in section 4.

3.1 Organization of On-chip Prefetcher

Traditional prefetchers actively fetch data from lower level memories to higher level memories before it is requested by the processor. In our study, the prefetch buffer, which holds the prefetched data from the slower memory (PCM in this case) sits on the core-side global prefetch controller (GPC), as shown in the Figure 1. We assume the prefetch buffer is SRAM. Since such on-chip storage is a scarce resource we only prioritize prefetching from the slower memory (PCM). In the Figure 1, for better visibility we are showing only one memory channel for each type of memories, actually there are more of them. We assume there is one memory controller per channel. The prefetch policy engine (PPE) implements the necessary prefetching policy and sends the prefetch requests to the PCM memory controller. The prefetch queue (Pref. Q) sitting inside the PCM memory controller, holds the prefetch requests for the PCM channel. The prefetching path is shown in Figure 1 by the dotted path 1 and 2. On a demand memory request, the prefetch buffer is checked first and on a prefetch buffer hit, the data is moved to the LLC as shown in Figure 1 by solid path 1 and 2. On a prefetch buffer miss, the request goes to the read queue (RDQ) for a memory read and finally the request is served by the memory channel, this path is shown in Figure 1 by solid path 1, 3, and 4.

3.2 Distance Prefetching

We implement the distance prefetching policy [14] which is a correlation based policy and a generalized form of Markov prefetching [12]. The general idea is to look at the past history of deltas (differences) between memory addresses (those that miss the LLC) to predict the future addresses and prefetch them. Distance prefetching uses a correlation table with one entry for each unique delta between any two consecutive miss addresses. This delta serves as the index into this table. Each entry stores the history of deltas seen right after the current delta. In our study, we explored distance prefetching since it is more generalized than Markov preteching, does not depend on the PC value and is useful in detecting repeating sequences even with non-unit strides. We used the Global History Buffer (GHB) structure to implement the distance prefetcher as presented in Nesbit *et al.* [25] and call it Global Delta Correlation (GDC) prefetching policy. In the GDC prefetching policy, the *width* degree implies how many different paths we want to explore and the *depth* degree suggests how far into future we want to prefetch; more details are available in Nesbit et al [25]. For our core-side prefetcher we evaluated GDC policy with *width* degree 1 and *depth* degree 4. We choose the GHB method for implementation since it requires smaller storage, provides higher accuracy and fewer conflicts as compared to the other table based implementations. For example, for a 512 entry Index table and a 512 entry GHB table the storage overhead is only 8KB [25].

3.3 CAMEO

One state of the art two level memory management study by Chou *et al.* proposes CAMEO (CAche-like MEmory Organization) [4], which integrates 3D-stacked DRAM (faster memory) and commodity DRAM (slower memory) in a hybrid main memory system. In CAMEO both the faster and slower memory are visible to the OS as flat address space, increasing the total physical memory capacity and also the faster memory serves as a hardware cache providing faster access to the recently accessed data. The cache works with conventional cache line size (64 byte) granularity, exploiting the data locality on a fine-grained basis. When a cache line is requested from the slower memory, CAMEO swaps the requested cache line with another cache line in the faster memory allowing subsequent request to the same cache line being serviced by a low latency and high bandwidth access to the faster memory. This swapping process ensures that there is only one copy of the line in the entire memory and hence the memory capacity is maximized. The swapping mechanism in CAMEO is handled by a table *(Line Location Table or LLT)* which tracks the physical location of the data lines. The advantage of CAMEO over related page migration policies lies in the simplicity of operation; it relies on well-known cache policies and does not require access count mechanisms of other related work [21, 34].

In this study, we want to evaluate the performance of CAMEO for heterogeneous memory systems involving NVMs. We employ a heterogeneous memory system involving HBM and PCM in a CAMEO like model, where HBM serves as the faster memory and PCM serves as the slower memory.

4. EXPERIMENTAL SETUP

Processor	**Values**
Number of cores	16
Core frequency	3.2 GHz
Issue width	4-wide Out-of-Order
ROB size	128 entries
Caches	**Values**
L1 I-cache (private)	32 KB, 2-way set associative
L1 D-cache (private)	16 KB, 4-way set associative
L2 cache (shared)	16 MB, 16-way set associative
PCM Memory	**Values**
Channels, capacity	2, 16 GB (2 x 8 GB)
Memory Controller (MC)	1 per channel
Ranks, banks	1 rank/channel, 8 banks/rank
Row buffer size	2 KB
Read queue	64 entries/MC
Write queue	256 entries/MC
Prefetch queue	32 entries/MC
Read latency	80 ns (6ns tPRE + 69ns tSENSE + 5ns tBUS)
Write latency	250 ns tCWL
Bus (per channel)	64-bit, 400MHz (LPDDR 800MHz)
HBM Memory	**Values**
Channels, capacity	8, 1 GB (8 x 128 MB)
Memory Controller (MC)	1 per channel
Ranks, banks	1 rank/channel, 2 bank groups/rank, 4 banks/bank group
Row buffer size	2 KB
Read queue	32 entries/MC
Write queue	32 entries/MC
tCAS-tRCD-tRP-tRAS	14 ns - 14 ns - 14 ns - 34 ns
Bus (per channel)	128-bit, 500MHz (DDR 1.0 GHz)

Table 1: Baseline configuration

4.1 Simulation Infrastructure

We use Ramulator [17] to simulate a 16-core system with main memory comprised of 1 GB HBM and 16 GB PCM. Detailed configuration parameters are shown in Table 1. We follow [24] for PCM latency parameters. The simulator is used in trace-driven mode with a CPU model to estimate instruction-per-cycle (IPC). The traces were generated using PinPlay kit [10] to identify and capture load/stores from a region of interest (ROI) of one billion instructions for each of the benchmarks in the case of single-threaded

No.	Workload	Benchmarks	MPKI	Footprint (GB)
1	mcf	16x mcf	65.04	16.03
2	lbm	16x lbm	44.21	6.30
3	milc	16x milc	23.05	9.05
4	omnetpp	16x omnetpp	18.96	2.06
5	astar	16x astar	16.80	2.63
6	GemsFDTD	16x GemsFDTD	9.59	10.59
7	zeusmp	16x zeusmp	8.14	3.32
8	bwaves	16x bwaves	6.90	6.82
9	cactusADM	16x cactusADM	3.70	2.31
10	xalancbmk	16x xalancbmk	4.50	2.89
11	mix1	mcf-sphinx-astar-lbm-gcc-soplex-mcf-libquantum -lbm-soplex-astar-milc-milc-mcf-omnetpp-libquantum	29.36	5.64
12	mix2	lbm-mcf-dealII-soplex-dealII-bzip2-cactusADM-soplex -GemsFDTD-soplex-lbm-lbm-GemsFDTD-mcf-cactusADM-dealII	20.47	5.08
13	mix3	GemsFDTD-libquantum-milc-dealII-sphinx-leslie3d-cactusADM -gcc-bzip2-sphinx-leslie3d-cactusADM-GemsFDTD-astar-gcc-milc	10.99	3.34
14	mix4	mcf-libquantum-soplex-GemsFDTD-milc-leslie3d-lbm-gcc-bzip2 -soplex-cactusADM-dealII-soplex-libquantum-libquantum-bzip2	18.27	3.60
15	mix5	mcf-lbm-soplex-lbm-mcf-soplex-mcf-mcf -soplex-lbm-soplex-soplex-lbm-lbm-mcf-lbm	42.51	7.61
16	mix6	libquantum-omnetpp-gcc-omnetpp-sphinx-milc-libquantum-gcc -libquantum-sphinx-sphinx-astar-libquantum-milc-gcc-omnetpp	19.64	2.11
17	xsbench	XSBench multi-threaded (16 threads)	22.01	14.68
18	lulesh	LULESH multi-threaded (16 threads)	13.51	6.80
19	miniFE	16x miniFE	6.72	10.66
20	CoMD	16x CoMD	1.41	2.30

Table 2: Evaluated workloads

benchmarks or for each of the threads in the case of multi-threaded benchmarks. Then we generate a 16-core multi-programmed/multi-threaded memory access trace (details are provided in section 4.2) using the multicore cache simulator Moola [33]. In our evaluation, we use hybrid memory system as shown in Table 1 without any prefetching and any data migration as our baseline.

4.2 Workloads

We choose 17 memory-intensive benchmarks from the SPEC CPU2006 suite [35]. Additionally, we used four HPC benchmarks, XSBench [2], LULESH [18], CoMD [23] and miniFE [7] which are representative benchmarks from the US Department of Energy (DOE) for evaluating HPC systems [1]. As shown in Table 2, there are 20 different workloads made up by mixing applications from the above mentioned benchmarks. To generate a multi-programmed workload (1-16, 19, and 20 in Table 2) we take 16 ROI traces (16 copies of traces of one application or traces from different applications) and bind each trace to one core of Moola cache simulator and generate a 16-core multi-programmed memory access trace that can then be used by Ramulator. To generate a multi-threaded workload (17 and 18 in Table 2) we take ROI traces from each of the threads of a 16-threaded application and similarly use Moola to generate a 16-core multi-threaded memory access trace that can then be used by Ramulator.

5. EVALUATION

For all of our experiments, the baseline is a hybrid memory system as shown in Table 1 without any prefetching or migration. We first analyze the prefetching policies in section 5.1. In the next section 5.2 we compare the performance improvement provided by our best core-side prefetching policy with CAMEO. In section 5.3 we show the Energy Delay Product (EDP) analysis of the techniques discussed in section 5.2. We present the prefetch buffer capacity sensitivity and PCM access latency sensitivity studies in section 5.4.

5.1 Prefetching Policy Analysis

We assume the on-chip prefetch buffer is a SRAM with 2 MB capacity organized as a 16-way set associative, write-back with least recently used (LRU) eviction policy buffer. After performing capacity sensitivity analysis for on-chip buffer sizes (1/2/4 MB sizes as presented in section 5.4), we found that 2 MB is reasonable since it is small enough to be placed on-chip and it provides similar performance improvement as a 4 MB buffer. We evaluate GDC prefetching with two different granularities- conventional cache line size granularity (64 bytes) and main memory row buffer size granularity (2 KB, we simply refer to it as block granularity). In case of block granularity (2 KB) prefetching, for a miss address we take the block address containing the cache line instead of the cache line address. Hence, we use Distance prefetching to predict the block sequence and a similar approach was taken in [27].

It is important to note that we prefetch only from the slower memory (in our case which is PCM). We adopt this approach since HBM is already much faster and provide higher bandwidth than PCM hence we do not want the data lines/blocks prefetched from PCM to be evicted by the data lines/blocks coming from HBM. In other words, we do not want to introduce thrashing like situation in the much

Policy Configuration	
Legend in Figure	**Description**
GDC_64B	GDC policy, width degree=1 and depth degree=4, 64 byte prefetch granularity
GDC_2KB	GDC policy, width degree=1 and depth degree=4, 2 KB prefetch granularity

Table 3: Experimental configurations of prefetching policies

smaller prefetch buffer by prefetching from both faster and slower memory. Details of the policies are provided in Table 3. These prefetching policies are hardware implemented and they do not require the load/store PC (or instruction pointer-IP) information to generate prefetch requests.

Figure 2 presents the IPC improvement (in percentages) provided by different core-side prefetching policies over the baseline without any prefetching/migration using the left y-axis. The positive y-axis on the left shows IPC improvement whereas the negative y-axis on the left shows performance degradation. The GDC_2KB scheme provides the best average IPC improvement of 15%, whereas GDC_64B provides 8% average IPC improvement. The result shows that prefetching at larger granularity provides better performance improvement. To analyze the result further we use two different metrics- Accuracy and Coverage to evaluate the prefetching policies. Following [19] we define the metrics below-

$$\mathit{Accuracy} = \mathit{Num\_of\_useful\_prefetches} \div \mathit{Num\_of\_prefetches}$$

$$\mathit{Coverage} = \mathit{Num\_of\_prefetch\_hits} \div (\mathit{Num\_of\_demand\_requests\_to\_PCM} + \mathit{Num\_of\_prefetch\_hits})$$

Note that we calculate the Coverage metric with respect to the demand memory requests going to PCM only, since we do not prefetch from HBM. Figure 3 shows the Accuracy (in percentages) and Figure 4 shows the Coverage (in percentages) of the prefetching policies. For the most workloads we observe that better prefetch Accuracy and Coverage together lead to better IPC improvement as expected. For three of the workloads (omnetpp, astar and xalancbmk) we observe exceptions that GDC_64B policy has the best Accuracy and Coverage but it does not provide better performance than GDC_2KB policy. We further investigate on this and found that since the GDC_2KB policy prefetches in 2KB granularity it generates memory requests to contiguous memory addresses which fall in the same row buffers (actually two row buffers over two PCM memory channels since we use cache line level address interleaving over the channels), hence it achieves higher row buffer hits for prefetch requests and can prefetch faster than GDC_64B. On the other hand GDC_64B policy prefetches in cache line granularity and may send prefetch requests to a number of different row buffers, hence it has lower row buffer hits. Also, for these workloads, GDC_2KB introduces more row buffer hits even for demand requests than GDC_64B scheme. We next investigate lbm's performance degradation for GDC_64B policy. Though we prefetch opportunistically when there is no demand read request pending to the same bank, the existing prefetch request which is being serviced currently need to be completed first before the next demand request to the same bank can be serviced, hence the demand request may be delayed. We found that for lbm GDC_64B policy introduces 10x more row buffer conflicts for prefetch reads than GDC_2KB policy leading to performance degradation.

5.2 Performance Analysis

Figure 5 shows IPC performance improvements (in percentages) of core-side prefetching (GDC_2KB) policy and CAMEO over the baseline without any prefetching/migration. A positive y-axis on the left shows IPC improvement whereas the negative y-axis on the left shows degradation. The right y-axis shows CAMEO hit-rate in percentages for analysis purposes. The hit-rate represents the fraction of memory requests that were found in HBM over the total number of demand memory requests. The overall IPC improvements for CAMEO and core-side prefetching over the baseline are 50% and 15% respectively.

CAMEO-based migration technique outperforms the core-side prefetching for $2/3^{\text{rd}}$ of the workloads in this study as shown in Figure 5. For workloads (mcf, omnetpp, astar, mix1, mix2, mix3, mix5, and mix6) which have higher IPC improvements by CAMEO (at-least 54% over core-side prefetching and 70% over no-prefetching) also show higher CAMEO hit rates (over 74%). The six of the best performing (with respect to CAMEO) workloads (mcf, omnetpp, astar, mix1, mix5 and mix6) exhibit very low spatial locality (on logical 2 KB block granularity) and thus they adopt well with CAMEO since it leverages locality on cache line basis. On the other hand, for around $1/3^{\text{rd}}$ of the workloads (milc, GemsFDTD, zeusmp, bwaves, cactusADM, lulesh, and miniFE) which have lower CAMEO hit rates (~64% and below) show worse performance with CAMEO than the core-side prefetching. Lower hit rate in CAMEO leads to more swapping and hence degrades the performance since PCM is much slower. For these workloads core-side prefetching provides on average 19% IPC improvement over CAMEO. For zeusmp and bwaves, CAMEO even performs worse than no-prefetching baseline. We attribute this behavior to smaller workload footprint (in such cases the baseline already performs better since a significant portion of the footprint is resident in HBM), as can be seen with lower CAMEO hit rates (54% and 24% respectively for zeusmp and bwaves), higher miss-prediction rate and its associated latency penalty. The lower hit-rate of CAMEO for some applications may be due to the spatial locality of the workloads not exploited by CAMEO. However the temporal locality of all workloads is exploited as CAMEO works more like a cache storing frequently accessed data lines in the faster memory. CAMEO uses a direct-mapped HBM cache organization optimized with co-locating tags with the data lines, which minimize hit latency. Since direct-mapped cache has more conflict misses, this may be another reason for lower hit rates in HBM for some workloads. Also for core-side prefetching, in a few cases (mcf, astar, mix5, and xsbench) we observe very little (less than 5%) performance gains over the baseline. We attribute this to high (over 16) misses per kilo instruction (MPKI) of the workloads and not being prefetch friendly. With much larger prefetch buffer

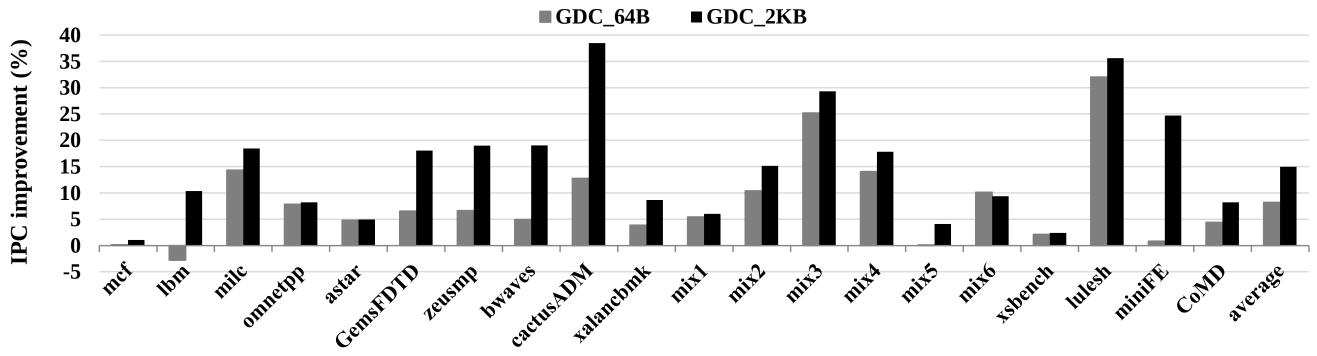

Figure 2: IPC improvement (%) of different prefetching policies over the baseline without any prefetching/migration

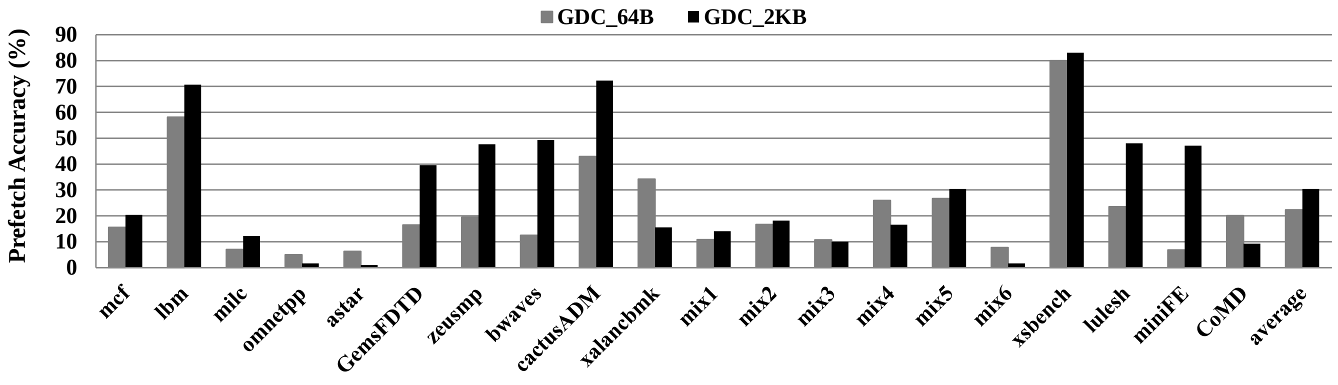

Figure 3: Accuracy (%) of different prefetching policies

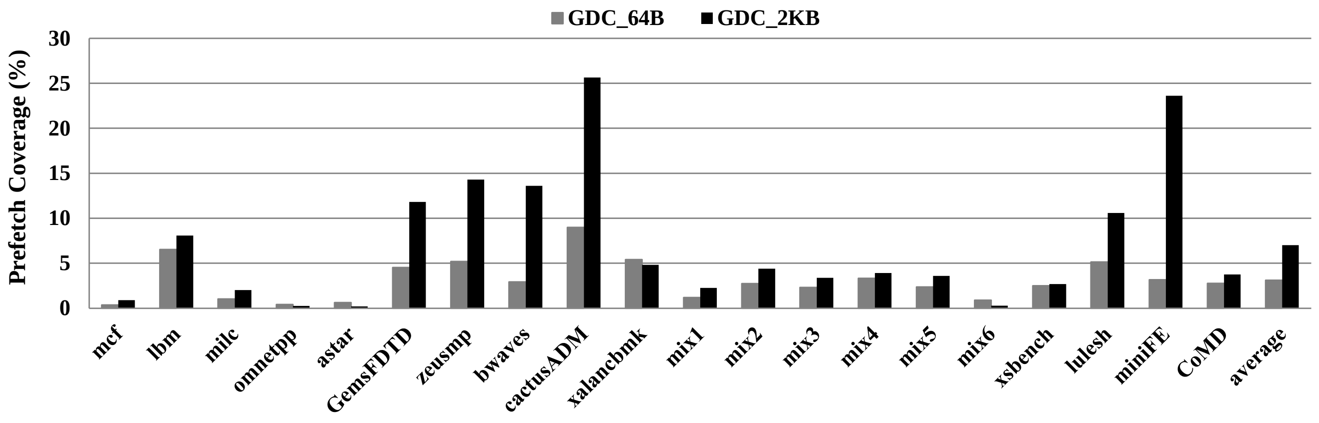

Figure 4: Coverage (%) provided by different prefetching policies

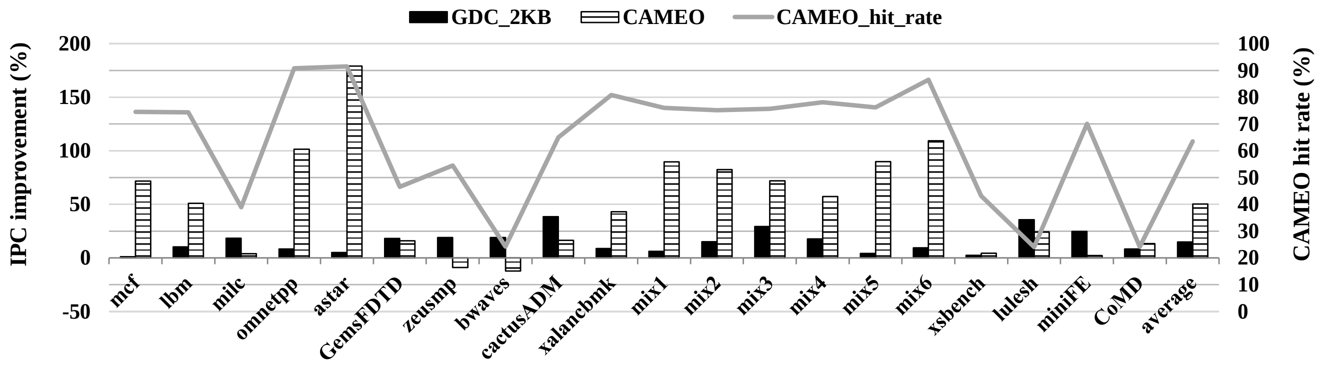

Figure 5: IPC improvement (%) provided by on-chip prefetching policy (GDC) and CAMEO over the baseline without any prefetching/migration (left y-axis) and CAMEO HBM hit rate (right y-axis)

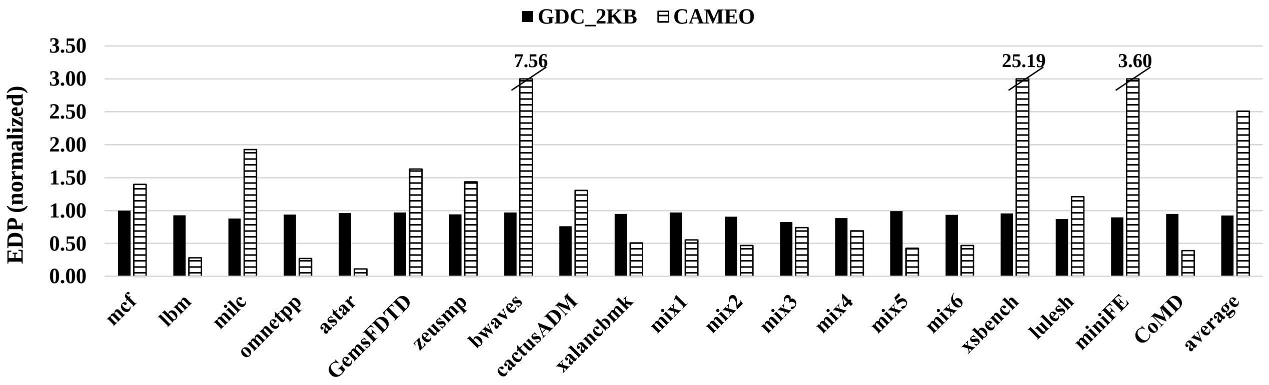

Figure 6: EDP of GDC_2KB prefetching policy and CAMEO normalized to the baseline without any prefetching/migration

and varying the prefetching policy might improve the results.

5.3 Energy Delay Product Analysis

We also analyze dynamic Energy Delay Product (EDP) for our workloads, in order to quantify energy versus performance trade-offs. Here we only considered dynamic memory energy. By multiplying the total energy consumed by memory accesses by the total execution time we get the EDP. Lower EDP is preferable. For SRAM implemented on-chip prefetch buffer access energy, we used values obtained from CACTI [37]. For PCM read/write energy values we used the numbers as presented in [39] and for HBM access energy value we relied on publicly available information in [28].

Figure 6 shows the EDP values for GDC_2KB prefetching policy and CAMEO, normalized with respect to the EDP values for the baseline without any prefetching/migration. The EDP values for the prefetching policy follows the trend of the performance data, as expected. Prefetching improves EDP for each of the workloads (in few cases by only 1%) and on average by 8% over the baseline. In the case of CAMEO, for every miss in HBM we need to read the data (in cache line granularity) from PCM into HBM (as HBM is a cache, it needs to be filled on every demand miss). Since HBM is a direct-mapped cache, to store a new cache line we have to evict the older cache line residing in the same set. This evicted cache line from HBM need to be written back to PCM even if it is clean (since HBM is also part of the hybrid physical memory address space and there is only one copy of the data in the entire physical memory). In other words, every HBM miss results in a write back to PCM. So for workloads which have higher miss rates, CAMEO EDP rises since PCM write consumes high energy. There are three extreme cases- bwaves, xsbench, and miniFE for which CAMEO EDP is much higher than the baseline system EDP. We investigated these cases and found that the actual workloads have fewer writes to PCM in the baseline configuration but in case of CAMEO they result in very large number of writes to PCM leading to much higher EDP. For example, in the baseline configuration for the xsbench workload only 0.1% of the total demand memory accesses are writes but in case of CAMEO 75% of the demand accesses result in writes into PCM, hence the CAMEO EDP is much higher than the baseline EDP in this particular case. We feel this is an important observation since for NVMs which have limited write endurance (e.g., PCM, RRAM [29]) CAMEO or any page migration policy may decrease the life time of such memories more rapidly than usual.

5.4 Sensitivity Analysis

5.4.1 Prefetch Buffer Capacity Sensitivity Analysis

Figure 7 shows IPC improvements for GDC_64B and GDC _2KB schemes with buffer capacities of 1, 2 and 4 MB. In the legends of Figure 7 we augment the policy name with the respective buffer size. For GDC_64B we generally see performance degradation going from smaller buffer size to larger buffer size. This is partially due to the increased tag array lookup delay for larger prefetch buffer with smaller prefetch granularity (64 byte). Since every demand miss to PCM is checked against the prefetch buffer tag array, a miss adds a delay overhead to each memory request that ends up going to PCM memory. To improve overall performance, prefetch Coverage should be high enough to hide the aforementioned delay factor introduced by prefetching. But we found that such increase in buffer size does not provide considerable improvement in prefetch Coverage. The prefetch buffer is a 16-way set associative buffer and we expect larger sized buffer should decrease the number of conflicts among the prefetched blocks and hence a prefetched block can retain in the buffer for longer period and provide more hits. But as we analyze the temporal locality behavior of the workloads we found that for most of the workloads the re-use distance of most of the accesses is larger than the time interval that the block can reside in the prefetch buffer, hence though we double the buffer size it is still small in absolute scale (up to only 4MB) which fails to provide much higher prefetch Coverage. For GDC_2KB policy we see negligible performance improvements for more than half of the workloads for larger buffer sizes. We note here that for GDC_2KB scheme the prefetch buffer tag array size is much smaller (∼32x smaller) than that of GDC_64B scheme since the latter prefetches in 64 byte granularity whereas the former prefetches in 2 KB granularity (thirty two 64 byte lines require one tag). Hence for GDC_2KB larger buffer size do not introduce substantial increment in tag array lookup time and we see slight performance improvement on average.

Timing Config. Name	**Description**
basic	PCM timing as presented in Table 1
fast	PCM read/write 2x faster than the basic PCM timing
slow	PCM read/write 2x slower than the basic PCM timing

Table 4: PCM timing configurations

5.4.2 PCM Access Latency Sensitivity Analysis

To evaluate the impact of PCM read/write timing on the aforementioned prefetching policies and CAMEO, we have repeated our experiments for a faster and a slower PCM memories as described in Table 4. Here we cover a timing range starting from a fictitious highly latency optimized PCM which may become available in future to much slower Multi-Level Cell (MLC) PCM. Figure 8 shows the PCM access latency sensitivity analysis for the three configurations shown in Table 4. In Figure 8, for each timing configuration (e.g., fast), IPC improvements (degradations) are shown with respect to the IPC of no-prefetching in that particular timing configuration (e.g., fast). The positive y-axis on the left shows IPC improvement whereas the negative y-axis on the left shows performance degradation.

In section 5.2 we found that for around 1/3rd of the workloads (milc, GemsFDTD, zeusmp, bwaves, cactusADM, lulesh, and miniFE), GDC_2KB policy provides better performance than CAMEO. The observation still holds for all the timing configurations with a few exceptions, CAMEO now outperforms GDC_2KB for milc, GemsFDTD, and lulesh for the fast timing and for miniFE for the slow timing. For fast PCM the amount of time we save by getting a prefetch buffer hit or by HBM hit in case of CAMEO is smaller than the time we save in case of slower PCMs, hence we observe overall smaller gain with fast PCM. IPC gains/losses (over baseline in respective timing configuration) provided by CAMEO

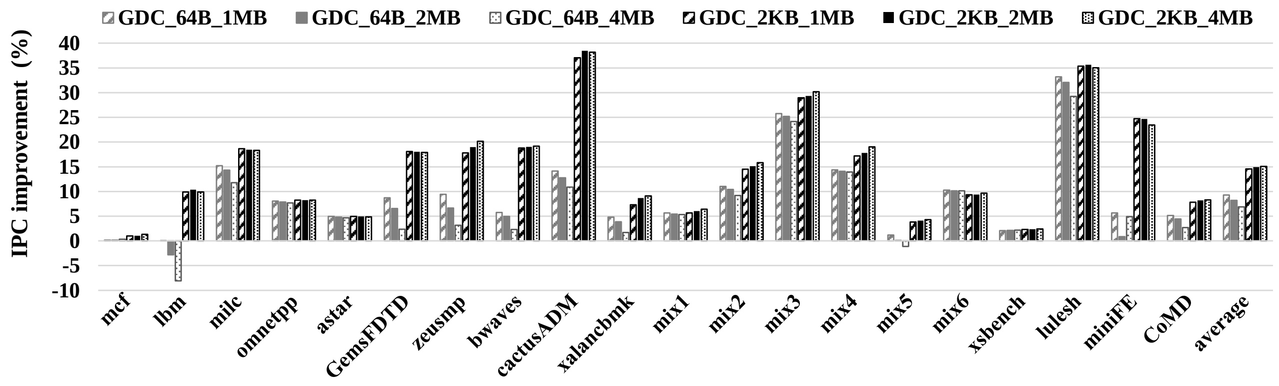

Figure 7: Prefetch buffer capacity sensitivity analysis for different prefetching policies. Prefetch buffers with capacities 1 MB, 2 MB and 4 MB are evaluated.

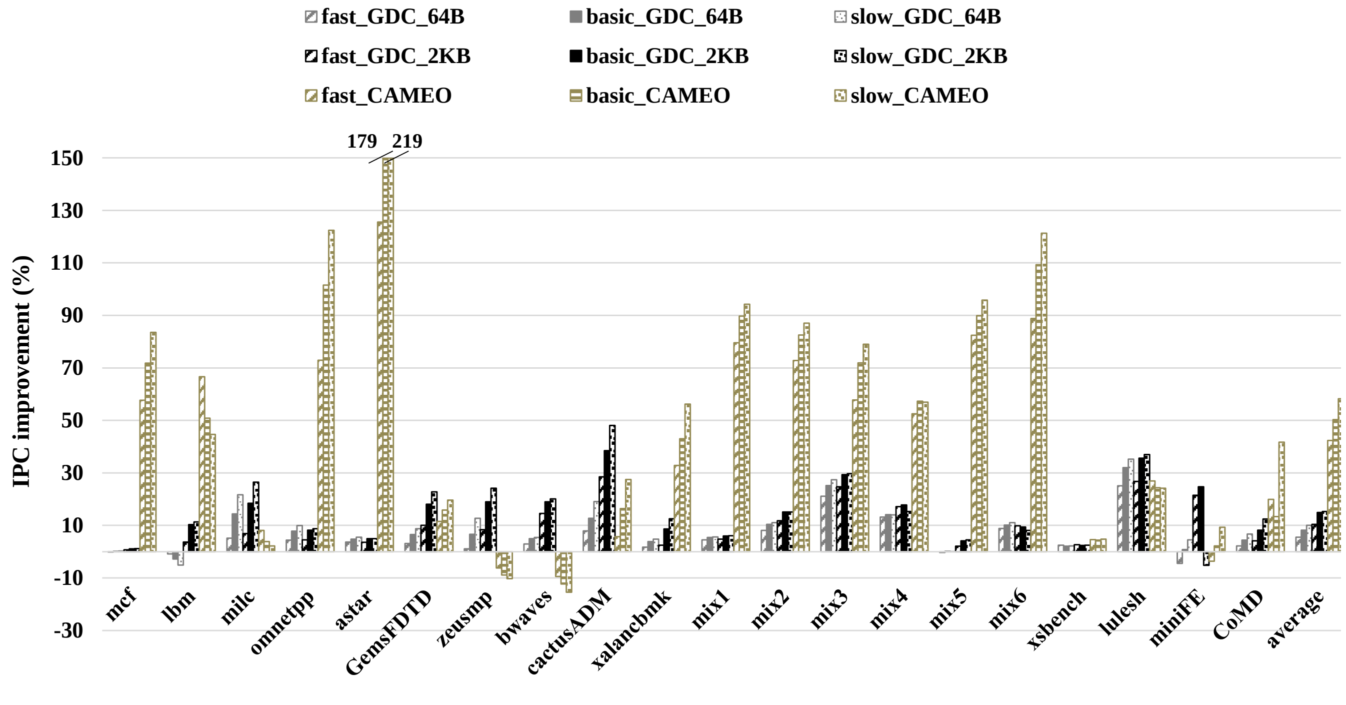

Figure 8: PCM access latency sensitivity analysis for different prefetching policies and CAMEO. Three different PCM timing configurations- fast, basic, and slow are evaluated. For each timing configuration (e.g., fast), IPC improvements (degradations) are shown with respect to the baseline IPC in that particular timing configuration (e.g., fast).

keeps increasing for slower PCM for all of the workloads except five of them (lbm, milc, mix4, xsbench, and lulesh). For these five workloads IPC gains are decreased for slower PCMs (note that there are still IPC improvements, only the gains are lower relative to the faster PCMs). We found that for most of these workloads PCM row buffer hit rates are very high (at least 87%) and hence the impact of slower PCM (increased read time from PCM data array to row buffer) is not too severe in the baseline case. However, as CAMEO introduces many row buffer conflicts in PCM for these workloads, for much slower PCM it incurs a delay factor and hence the attained IPC improvements are decreased. The improvement rate of GDC_2KB policy keeps improving over no-prefetching for slower PCM for all of the workloads except five of them (mix2, mix4, mix6, xsbench, and miniFE). For workloads mix2, mix4, mix6, and xsbench we see that prefetch Accuracy and Coverage decrease for slower PCMs and hence there are smaller performance improvements. For miniFE though the prefetch Accuracy and Coverage of GDC_2KB are high for the slow timing configuration PCM, we found that, it introduced large number of demand read row buffer conflicts, which led to overall performance degradation. For GDC_64B policy we see that IPC gains/losses (over baseline in respective timing configuration) keep increasing monotonically for slower PCM for most of the workloads.

6. CONCLUSION AND FUTURE WORK

When PCM is employed as a part of system main memory, our evaluations show that the CAMEO-based data migration leads to performance improvements over our core-side prefetching for 2/3rd of the workloads but for 1/3rd of the workloads the performance is worse than core-side prefetching and in a few cases even worse than no-prefetching. The lower performance improvement (or degradation) provided by CAMEO is due to the large number of accesses to the slower memory (viz., PCM) in case of lower CAMEO hit rate. A second insight of our evaluation is that for NVM (e.g., PCM), CAMEO may cause write endurance issues due to frequent swaps specially when the hit rates are lower and this may also lead to high energy consumption. When PCM is used as a part of main memory, prefetching seems to be a better choice for some workloads. Also, the prefetching policy evaluated in this work is a simple implementation but it can further be optimized to get much higher performance improvements for PCM-based hybrid memory systems. However, CAMEO-based migration seems to perform better for several benchmarks, thus it may be worthwhile combining prefetching and migration.

As a continuation of this work, we plan to explore additional prefetching policies which will be better suited for multi-level hybrid memory systems than the conventional prefetching policies. We also plan to combine both prefetching and migration in the same hybrid memory system. We believe such combined approach would be beneficial when a part of the hybrid memory is made up of NVM which generally comes with write-related limitations.

7. ACKNOWLEDGMENT

We would like to sincerely thank Marko Scrbak for his help in reviewing the paper draft. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. This work is also supported in part by the NSF Net-Centric Industry/University Cooperative Research Center.

8. REFERENCES

[1] Co-Design | U.S. DOE Office of Science (SC). http://science.energy.gov/ascr/research/scidac/co-design/.

[2] Proxy-Apps for Neutronics. https://cesar.mcs.anl.gov/content/software/neutronics.

[3] J. Ahn, S. Yoo, and K. Choi. Low-power hybrid memory cubes with link power management and two-level prefetching.

[4] C. Chou, A. Jaleel, and M. K. Qureshi. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. IEEE Computer Society, 2014.

[5] ExtremeTech. PCM. http://www.extremetech.com/extreme/182096-ibm-demonstrates-next-gen-phase-change-memory-thats-up-2015.

[6] J. W. Fu, J. H. Patel, and B. L. Janssens. Stride directed prefetching in scalar processors. *ACM SIGMICRO Newsletter*, 23(1-2):102–110, 1992.

[7] M. Heroux and S. Hammond. MiniFE: Finite Element solver. https://portal.nersc.gov/project/CAL/designforward.htm#MiniFE.

[8] Intel. KnightsLanding. http://www.realworldtech.com/knights-landing-details/.

[9] Intel. Fun Facts: How Fast and Robust is 3D XPoint Technology? http://www.intel.com/newsroom/kits/nvm/3dxpoint/pdfs/NextGen_NVM_FunFacts.pdf, 2015.

[10] Intel. PinPlay. https://software.intel.com/en-us/articles/program-recordreplay-toolkit, 2015.

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

[12] D. Joseph and D. Grunwald. Prefetching using markov predictors. In *ACM SIGARCH Computer Architecture News*, volume 25, pages 252–263. ACM, 1997.

[13] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, pages 364–373. IEEE, 1990.

[14] G. B. Kandiraju and A. Sivasubramaniam. *Going the distance for TLB prefetching: an application-driven study*, volume 30. IEEE Computer Society, 2002.

[15] K. Kavi, S. Pianelli, G. Pisano, G. Regina, and M. Ignatowski. Memory organizations for 3d-drams and pcms in processor memory hierarchy. *Journal of Systems Architecture*, 61(10):539–552, 2015.

[16] J. Kim and Y. Kim. HBM: Memory Solution for Bandwidth-Hungry Processors. *Hot Chips: A Symposium on High Performance Chips (HC26)*, 2014.

[17] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. 2015.

[18] Lawrence Livermore National Laboratory. Hydrodynamics Challenge Problem. Technical Report LLNL-TR-490254.

[19] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt. Prefetch-aware dram controllers. In *Proceedings of the 41st Annual IEEE/ACM international Symposium on Microarchitecture*, pages 200–209. IEEE Computer Society, 2008.

[20] J. Macri. AMD's Next Generation GPU and High Bandwidth Memory Architecture: FURY.

[21] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 126–136. IEEE, 2015.

[22] Micron. NVDIMM. https://www.micron.com/products/dram-modules/nvdimm#/, 2015.

[23] J. Mohd-Yusof, S. Swaminarayan, and T. C. Germann. Co-design for molecular dynamics: An exascale proxy application, 2013.

[24] P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi. Reducing read latency of phase change memory via early read and turbo read. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 309–319. IEEE, 2015.

[25] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *Software, IEE Proceedings-*, pages 96–96. IEEE, 2004.

[26] NVIDIA. NVIDIA Pascal. http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/.

[27] M. Oskin and G. H. Loh. A software-managed approach to die-stacked dram.

[28] J. T. Pawlowski. Hybrid memory cube (hmc). In *Hot Chips*, volume 23, 2011.

[29] M. K. Qureshi, S. Gurumurthi, and B. Rajendran. Phase Change Memory: From devices to Systems. *Synthesis Lectures on Computer Architecture*, 6(4):1–134, 2011.

[30] 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 *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 235–246. IEEE Computer Society, 2012.

[31] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. *ACM SIGARCH Computer Architecture News*, 37(3):24–33, 2009.

[32] S. Raoux, F. Xiong, M. Wuttig, and E. Pop. Phase change materials and phase change memory. *MRS Bulletin*, 39(08):703–710, 2014.

[33] C. F. Shelor and K. M. Kavi. Moola: Multicore Cache Simulator. *International Conference on Computers and Their Applications*, 2015.

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

[35] spec. SPEC CPU 2006. https://www.spec.org/cpu2006/, 2015.

[36] C. Su, D. Roberts, E. A. León, K. W. Cameron, B. R. de Supinski, G. H. Loh, and D. S. Nikolopoulos. Hpmc: An energy-aware management system of multi-level memory architectures. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 167–178. ACM, 2015.

[37] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. Technical report, Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[38] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu. Row buffer locality aware caching policies for hybrid memories. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 337–344. IEEE, 2012.

[39] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu. Efficient data mapping and buffering techniques for multilevel cell phase-change memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(4):40, 2015.