Analytical Study on Bandwidth Efficiency of Heterogeneous Memory Systems

Amin Farmahini-Farahani, David Roberts, Nuwan Jayasena
AMD Research, Advanced Micro Devices, Inc.
{afarmahi, david.roberts, nuwan.jayasena}@amd.com

Abstract
Heterogeneous memory systems integrate different memory technologies to balance design requirements such as bandwidth, capacity, and cost. Performance of these systems depends heavily on memory hierarchy organization, memory attributes, and application characteristics. In this paper, we present analytical bandwidth models for a range of heterogeneous memory systems composed of DRAM and non-volatile memory (NVM). Our models enable exploring heterogeneous memory systems with different organizations and attributes. Using the models, we study the bandwidth efficiency of heterogeneous memory systems to provide insights into the bandwidth bottlenecks of these systems under different application characteristics. Our analytical results highlight the importance of NVM read-write bandwidth asymmetry and DRAM-NVM bandwidth asymmetry in bandwidth efficiency. Specifically, in flat non-uniform memory access (NUMA) systems, the read bandwidth is maximized when a certain portion of bandwidth is delivered by DRAM and that portion depends on multiple factors including DRAM and NVM bandwidth attributes and application bandwidth characteristics. In DRAM-cache-based systems, when the hit rate is low, the impact of the DRAM cache organization on the read bandwidth is minimal. However, at higher hit rates and NVM bandwidths, the impact of the cache organization on sustained read bandwidth becomes pronounced.

CCS Concepts
•Computer systems organization → Heterogeneous (hybrid) systems; •Hardware → Memory and dense storage; •Hardware → Non-volatile memory; •Software and its engineering → Main memory;

1. Introduction

The emergence of a variety of new memory technologies is likely to fundamentally alter memory system design. On the one hand, new DRAM interface standards based on technologies such as high-speed serial links and 3D die-stacking (e.g., Hybrid Memory Cube (HMC) [15, 8, 2], JEDEC High Bandwidth Memory [1]) promise greatly enhanced memory bandwidths even as mainstream DDR-based DRAM continues to improve performance. On the other hand, new emerging non-volatile memory (NVM) technologies such as Resistive RAM and Phase Change Memory (PCM) promise greater cell densities that can provide increased memory capacity albeit typically at lower performance levels than DRAM-based technologies. Heterogeneous memory systems incorporating both DRAM and NVM hold the promise of higher bandwidth as well as higher capacity than is possible with either one of these classes of memories alone.

An example of a heterogeneous memory system is a flat-address non-uniform memory access (NUMA) memory system where high-performance DRAM and high-capacity NVM are exposed as regions of the same physical address space. The overall bandwidth of such a system depends heavily on efficient data placement and migration within the memories by software to keep frequently-accessed data in DRAM. Another example of a heterogeneous memory system is one where DRAM is used as a hardware-managed cache for NVM. The bandwidth efficiency of such a system is highly dependent on factors such as cache hit rate and tag storage design. Further, the performance of both these examples depends heavily on the NVM bandwidth.

In this paper, we describe a variety of heterogeneous memory system organizations consisting of high-bandwidth memory with limited capacity and low-bandwidth memory with higher capacity and develop comprehensive analytical models to compare their bandwidth efficiencies and utilizations under different configurations.

The rest of the paper is organized as follows. Section 2 introduces a bandwidth model for homogeneous memory systems. Section 3 describes our bandwidth models for a variety of heterogeneous memory systems and studies their bandwidth efficiencies. Section 4 discusses our analytical bandwidth results. Section 5 presents the latency breakdown of memory accesses in heterogeneous memory systems. Section 6 provides an overview of related work. Finally, Section 7 concludes the paper.

2. Bandwidth Model for Homogeneous Memory Systems

Memory bandwidth often dictates the execution time of memory-intensive applications. In this section, we study the bandwidth of memory-intensive applications in homogeneous

MEMSYS '16, October 03-06, 2016, Alexandria, VA, USA
 ISBN 978-1-4503-4305-3/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2989081.2989089

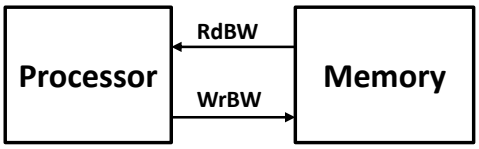

Figure 1. Logical view of a homogeneous memory system.

memory systems, and we then extend our study to heterogeneous memories in subsequent sections.

Figure 1 shows a logical view of a homogeneous memory system. The execution time (performance) of a memory-bandwidth-bound applications can be formulated as the following:

$$Execution\ time = Data\_Volume / BW \quad (1)$$

where *Data_Volume* is the total main memory data volume that is accessed during the application's execution and *BW* is the sustained bandwidth delivered by the main memory.

The main memory bandwidth can further be refined by separating memory read bandwidth from memory write bandwidth. In current volatile (e.g., DDR DRAM) and non-volatile (e.g., PCM) memory technologies, read and write bandwidth is shared both locally within a memory bank and externally over a single memory I/O interface, meaning that at any given time, either a read or a write can be performed. However, DRAM has symmetric read and write bandwidths, while NVM has asymmetric bandwidth characteristics where its write bandwidth is typically smaller than its read bandwidth.

In our study, we represent applications by their memory read and write bandwidth requirements as reflected by the traffic between the processor's on-chip last-level cache (LLC) and main memory. The amount of read and write bandwidth that an application consumes is a function of application characteristics (e.g., arithmetic intensity, memory access pattern) as well as compute-unit characteristics (e.g., number of cores and functional units, operating frequency, cache hierarchy characteristics). To refine our bandwidth model, we define some application bandwidth parameters in Table 1.

App_RdBW, *App_WrBW*, and *App_R2W* are defined as properties of the application and are independent of the physical properties of memory. Although *App_RdBW* and *App_WrBW* are application bandwidth parameters (with a unit of bytes/sec), they roughly resemble "program balance" between LLC and memory in the balance model [5] or the inverse of "operational intensity" in the roofline model [21].

Table 2 defines some memory parameters for the steady-state bandwidth potential of off-chip memory. The maximum sustained memory bandwidth (*Mem_RdBW* and *Mem_WrBW*) is lower than the maximum theoretical bandwidth of memory. For CPUs, the sustained bandwidth is typically 65-75% of the maximum theoretical memory bandwidth [18]. For example, a DDR3-1600 rank has a maximum theoretical bandwidth of 12.8GB/s which delivers a sustained bandwidth of about 9GB/s. For GPUs, the sustained bandwidth can be as high as 90-95% of the maximum theoretical memory bandwidth. *Mem_RdBW* and *Mem_WrBW* roughly resemble the "machine balance" between LLC and memory [5].

For clarity later in the paper, we define the maximum sustained read bandwidth of DRAM and NVM as *DRAM_RdBW* and *NVM_RdBW* and the maximum sustained write bandwidth of DRAM and NVM as *DRAM_WrBW* and *NVM_WrBW*. These parameters are properties of memory and are independent of application characteristics. In DRAM, *DRAM_RdBW* and *DRAM_WrBW* are equal as DRAM has symmetric read and write bandwidths, while in NVM *NVM_RdBW* is usually higher than *NVN_WrBW* [22, 23, 16] and it could be as high as 20X. Hence, the memory read to write bandwidth ratio for DRAM (*DRAM_R2W*) has a value of 1, while *NVM_R2W* usually has a value greater than 1.

Using the parameters defined above, we find the bandwidth delivered by main memory (*RdBW_Delivered* and *WrBW_Delivered*) in a homogeneous memory system for an application (shown in Figure 1). The relationship between delivered read and write bandwidths can be expressed as:

$$RdBW\_Delivered / WrBW\_Delivered = App\_R2W \quad (4)$$

Table 1. Application parameters

App_RdBW	Application read bandwidth demand after the LLC in bytes per second
App_WrBW	Application write bandwidth demand after the LLC in bytes per second
App_R2W	Read bandwidth to write bandwidth ratio of the application (*App_RdBW / App_WrBW*)

Table 2. Memory parameters

Mem_RdBW (*DRAM_RdBW* and *NVM_RdBW*)	Maximum sustained read bandwidth of off-chip memory in the absence of write traffic
Mem_WrBW (*DRAM_WrBW* and *NVM_WrBW*)	Maximum sustained write bandwidth of off-chip memory in the absence of read traffic
Mem_R2W (*DRAM_R2W* and *NVM_R2W*)	Read bandwidth to write bandwidth ratio of off-chip memory (*Mem_RdBW / Mem_WrBW*). Typically, *DRAM_R2W*=1 and *NVM_R2W* > 1.

Table 3. Delivered read and write bandwidths in homogenous memory systems

$$RdBW\_Delivered = min\left(App\_RdBW, \frac{Mem\_RdBW}{1 + \frac{Mem\_R2W}{App\_R2W}}\right) \quad (2)$$

$$WrBW\_Delivered = min\left(App\_WrBW, \frac{Mem\_RdBW}{Mem\_R2W + App\_R2W}\right) \quad (3)$$

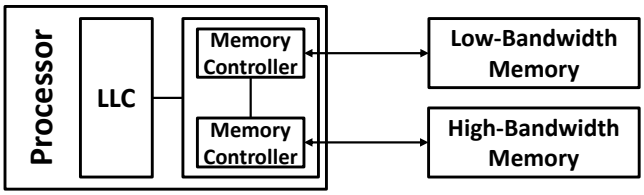

Figure 2. Physical structure of a heterogeneous memory system.

For memory-bandwidth-bound applications, the delivered read bandwidth is degraded from the maximum sustainable because writes consume some of that bandwidth. The fraction consumed by writes is inversely proportional to the memory system's read to write bandwidth ratio (*Mem_R2W*). Eq. (5) expresses this relationship.

$$Mem\_RdBW = RdBW\_Delivered + WrBW\_Delivered \times Mem\_R2W \quad (5)$$

For memory-bandwidth-bound applications, the bandwidth delivered by memory is independent of actual application bandwidth demand. For these applications, only the read bandwidth to write bandwidth ratio of the application is needed to calculate the delivered bandwidth.

On the other hand, for non-memory-bandwidth-bound applications, the application bandwidth demand (*App_RdBW* and *App_WrBW*) dictates the delivered bandwidth. Taking Equations (4)-(5) into account, we can derive expressions for *RdBW_Delivered* and *WrBW_Delivered* in Table 3.

In applications with multiple phases, the bandwidth delivered by memory varies as the application bandwidth demand could be different in each phase. Thus, the delivered bandwidth should be calculated separately for each phase.

3. Bandwidth Model for Heterogeneous Memory Systems

In this section, we model and evaluate the memory bandwidth efficiency of different heterogeneous memory systems composed of a high-bandwidth memory and a low-bandwidth memory. Figure 2 shows the physical structure of a heterogeneous memory system. While the memory controllers in this figure are shown implemented on the processor chip (as is the case with conventional DDR DRAM modules), alternate organizations where the controllers are integrated within the memory package (as in HMC) are also possible. There could also exist direct connections between the low-bandwidth memory and high-bandwidth memory (as in networks of memory nodes [19, 11]).

Heterogeneous memory systems can typically be categorized as flat NUMA and cache-based memory systems. In the flat NUMA memory system, data placement in the high-bandwidth memory and low-bandwidth memory is orchestrated by software. In the cache-based memory system, the high-bandwidth memory operates as a hardware-managed cache for the low-bandwidth memory. The system shown in Figure 2 can be organized as either a flat NUMA system as shown in Figure 3, or a cache-based system as shown in Figure 5.

For simplicity of description, we assume the high-bandwidth memory is made up of DRAM (and refer to it as DRAM) and the low-bandwidth memory is made up of NVM (and refer to it as NVM). However, our models can be used for other heterogeneous memory systems such as those composed of an on-package DRAM and off-package DRAM. We assume that the applications running on the processor exhibit high memory-level parallelism with a *uniform distribution of accesses* over the entire physical address space. We also assume the processor can support a large number of in-flight memory requests. The goal of our bandwidth models is to quantify the relative bandwidth efficiency of heterogeneous memory systems compared to a homogeneous memory system.

In the cache-based memory systems that we evaluate in this paper, DRAM cache is used as a write-back cache for NVM as this is the most common approach for DRAM caches and reduces write bandwidth. In these systems, DRAM cachelines are allocated on read misses and writebacks from the processor's LLC that miss in the cache. On a writeback to the DRAM cache, a full cacheline is written to DRAM from the processor's LLC and no data is read from NVM. This saves memory bandwidth for write-intensive workloads. Modified DRAM cachelines are written to NVM on cacheline replacement.

In order to capture the characteristics of heterogeneous memory systems, we define another parameter termed *Dram_HitRate*. For flat NUMA systems, it indicates what fraction of memory bandwidth is supplied by DRAM. For DRAM-cache-based systems, this parameter indicates what fraction of memory requests is filtered by the DRAM cache. We use the same hit rate for both reads and writes to make our models simpler. However, our models can be extended to have separate hit rates for reads and writes.

We also define another parameter termed *Dram_Dirty* which indicates the probability of a DRAM cacheline being dirty in systems with hardware-managed DRAM caches. To derive the *Dram_Dirty* expression, we make the observation that a clean line becomes dirty by a write (either hit or miss) and a dirty line becomes clean by eviction due to a read miss. In steady state, the ratio of dirty lines to total lines in the cache is constant. Eq. (6) defines *Dram_Dirty*. The derivation of this expression is described by Bolotin *et al.* [4].

$$Dram\_Dirty = \frac{1}{1 + App\_R2W \times (1 - Dram\_HitRate)} \quad (6)$$

3.1 Flat-address NUMA

In our first study of heterogeneous memory systems, we evaluate the bandwidth of flat NUMA memory systems. Figure 3 shows the logical view of a flat NUMA system. Using notations introduced in Figure 3, we first develop expressions for the bandwidth variables R_{DRAM}, R_{NVM}, W_{DRAM}, and W_{NVM} as functions of DRAM hit rate and application bandwidth demand. R_{DRAM} and W_{DRAM} define application read and write bandwidth demands on DRAM and R_{NVM} and W_{NVM} define

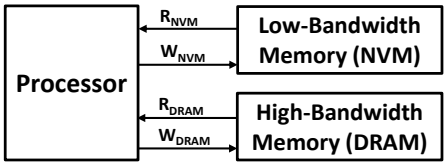

Figure 3. Logical view of the software-managed, flat-address NUMA memory system.

application bandwidth demands on NVM. In this model, we do not consider the bandwidth overhead of data movement between DRAM and NVM. Note that *Dram_HitRate* indicates the fraction of memory bandwidth supplied by DRAM (high-bandwidth memory).

$$R_{DRAM} = App\_RdBW \times Dram\_HitRate \quad (7)$$

$$R_{NVM} = App\_RdBW \times (1 - Dram\_HitRate) \quad (8)$$

$$W_{DRAM} = App\_WrBW \times Dram\_HitRate \quad (9)$$

$$W_{NVM} = App\_WrBW \times (1 - Dram\_HitRate) \quad (10)$$

In this memory organization, the bandwidth delivered by DRAM can be limited by the bandwidth delivered by NVM as NVM might not be able to supply enough bandwidth for the portion of data that is placed in NVM. Similarly, the bandwidth delivered by NVM can be limited by the bandwidth delivered by DRAM as DRAM might not be able to supply enough bandwidth for the data placed in DRAM. Hence, we can find the bandwidth delivered by DRAM and NVM by deriving from Equations (2) and (3) and factoring in the interdependency between the DRAM bandwidth and NVM bandwidth.

Table 4 summarizes a set of equations for the bandwidth delivered by DRAM and NVM in this organization. To find the delivered read and write bandwidths, we calculate the maximum numbers for *DRAM_RdBW_Delivered* and *NVM_RdBW_Delivered* by solving Equations (11)-(12) and maximum numbers for *DRAM_WrBW_Delivered* and *NVM_WrBW_Delivered* by solving Equations (13)-(14).

3.1.1 Optimal Data Placement

In this organization, data placement in DRAM and NVM determines the delivered bandwidths. An optimal data placement enables maximum utilization of both DRAM and NVM bandwidths concurrently. The total (both DRAM and NVM) read bandwidth is directly a function of the ratio of the delivered DRAM bandwidth and delivered NVM bandwidth. In other words, the total read bandwidth is maximized when a certain portion of memory bandwidth is supplied by DRAM and the rest by NVM. Eq. (15) defines the portion of memory bandwidth that should be supplied by DRAM in order to maximize the total read bandwidth. This equation considers asymmetry in NVM bandwidth as well as the application read to write ratio.

$$Optimal\_Dram\_HitRate = \frac{1}{1+\frac{NVM\_RdBW(1+App\_R2W)}{DRAM\_RdBW(App\_R2W+NVM\_R2W)}} \quad (15)$$

Eq. (15) also indicates the optimal data placement ratio in DRAM and NVM when data is accessed uniformly over the address space. When using NVMs with symmetric read and write bandwidths, this equation simplifies to $\frac{DRAM\_RdBW}{DRAM\_RdBW+NVM\_RdBW}$ which indicates that, when data is accessed uniformly, data should be placed in the bandwidth ratio of high-bandwidth memory and low-bandwidth memory to achieve the maximum total read bandwidth. This observation corroborates recent experimental results based on the heterogeneous memory systems of GPUs [3].

3.1.2 Bandwidth Efficiency and Utilization

We define bandwidth efficiency in heterogeneous memory systems as the total read bandwidth achieved from the application's perspective relative to the read bandwidth of a DRAM-only system. The bars in Figure 4 show bandwidth efficiency in this memory organization. The total read bandwidth achieved is the sum of the DRAM read bandwidth (R_{DRAM}) and NVM read bandwidth (R_{NVM}). In this figure, we assume DRAM has a sustained bandwidth of 19GB/s, which is similar to the bandwidth of a conventional DDR4 DRAM rank. We also assume the application is memory-bound with

Table 4. Delivered read and write bandwidths in DRAM+NVM heterogeneous memory systems with a flat NUMA organization

$$DRAM\_RdBW\_Delivered = \min\left(R_{DRAM}, \frac{Dram\_HitRate}{1-Dram\_HitRate} \times NVM\_RdBW\_Delivered, \frac{DRAM\_RdBW}{1+\frac{W_{DRAM}}{R_{DRAM}}}\right) \quad (11)$$

$$NVN\_RdBW\_Delivered = \min\left(R_{NVM}, \frac{1-Dram\_HitRate}{Dran\_HitRate} \times DRAM\_RdBW\_Delivered, \frac{NVM\_RdBW}{1+\frac{NVM\_R2W}{(R_{NVM}/W_{NVM})}}\right) \quad (12)$$

$$DRAM\_WrBW\_Delivered = \min\left(W_{DRAM}, \frac{Dram\_HitRate}{1-Dram\_HitRate} \times NVM\_WrBW\_Delivered, \frac{DRAM\_RdBW}{1+\frac{R_{DRAM}}{W_{DRAM}}}\right) \quad (13)$$

$$NVN\_WrBW\_Delivered = \min\left(W_{NVM}, \frac{1-Drm\_HitRate}{Dram\_HitRate} \times DRAM\_WrBW\_Delivered, \frac{NVM\_RdBW}{NVM\_R2W+\frac{R_{NVM}}{W_{NVM}}}\right) \quad (14)$$

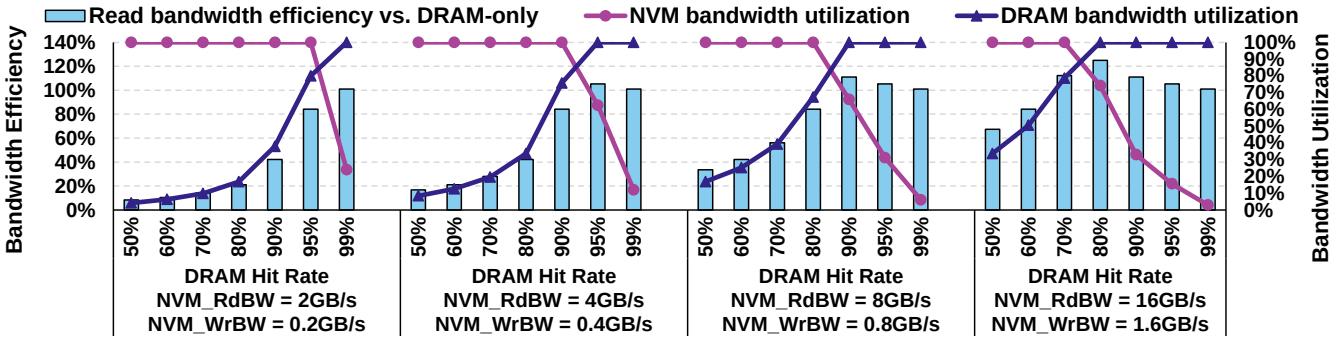

Figure 4. Bandwidth efficiency and utilization in DRAM+NVM heterogeneous memory systems with a flat NUMA organization (DRAM_RdBW = 19GB/s, App_R2W = 5).

a 5-to-1 read to write ratio after the processor's LLC (*App_R2W* = 5). The delivered read to write bandwidth ratio of both DRAM and NVM is also 5-to-1 as no data cache is used between the processor's LLC and memory (DRAM and NVM). For this DRAM and application configuration, the maximum achievable read bandwidth in a DRAM-only system is 15.83GB/s (5/6 × 19GB/s). In other words, a system with a read bandwidth efficiency of 100% delivers 15.83GB/s of read bandwidth. We vary the DRAM hit rate and NVM read and write bandwidth to explore the bandwidth efficiency of different configurations. The lines in Figure 4 show total (read and write) utilizations of DRAM and NVM bandwidths.

Figure 4 shows that bandwidth efficiency increases as DRAM hit rate increases up to a point and then tapers off. Bandwidth efficiency can even surpass 100% as data can be retrieved from both NVM and DRAM simultaneously, utilizing both bandwidths. Finding a sweet spot where both DRAM and NVM are not underutilized depends heavily on memory attributes and application characteristics. As shown by prior work, a judicious data placement enables efficient bandwidth utilization of both capacity-optimized memory and bandwidth-optimized memory [3]. Our model captures and explains this effect and incorporates additional parameters such as asymmetry in memory read and write bandwidths and the impact of application writes on bandwidth utilization.

The bandwidth efficiency is maximized when bandwidth utilization of both DRAM and NVM is 100%. When the DRAM hit rate exceeds the point at which DRAM utilization is maximized, NVM utilization decreases as DRAM cannot provide more bandwidth despite the increasing hit rate, causing NVM bandwidth underutilization.

For the configuration in Figure 4 (an application with a 5-to-1 read to write ratio) and NVM with 16GB/s read and 1.6GB/s write bandwidth, the maximum achievable bandwidth efficiency is 134%, meaning that DRAM and NVM provide 15.83GB/s and 5.33GB/s of read bandwidth, respectively. To achieve the maximum bandwidth efficiency in this configuration, 74.8% of read bandwidth should be supplied by DRAM ($Optimal\_Dram\_HitRate$) and the rest by NVM. This result highlights the importance of balancing bandwidth and avoiding bandwidth underutilization in flat NUMA memory organizations.

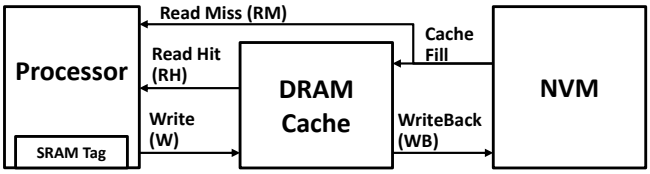

Figure 5. Logical view of the hardware-managed DRAM cache with SRAM tag storage.

3.2 DRAM Cache with SRAM Tag Storage

In our first study of hardware-managed DRAM cache systems, we assume tags for the DRAM cache are stored in the processor using SRAM storage as shown in Figure 5. In this system, no DRAM bandwidth is consumed on DRAM cache misses or for fetching tags as tags can be checked by the processor without sending any DRAM requests. This organization is typically impractical for high-capacity DRAM caches due to high tag storage overhead. Nonetheless, we study this organization to compare its bandwidth efficiency with the organizations in later sections that incur tag fetch overheads.

Before calculating the bandwidth in this organization, we analyze the read and write bandwidth delivered by the DRAM cache in Figure 5 *when the NVM bandwidth is not a limiting factor*. Using notations in Figure 5, *RH*, *RM*, *W*, and *WB* define read hit bandwidth, read miss bandwidth (which is the same as cache fill bandwidth), write bandwidth to DRAM cache, and writeback bandwidth from the DRAM cache to NVM, respectively. To study bandwidth delivered in this memory system, we develop expressions for the bandwidth variables *RH*, *RM*, *W*, and *WB* as a function of DRAM hit rate and application bandwidth demand.

$$RH = App\_RdBW \times Dram\_HitRate \quad (16)$$

$$RM = App\_RdBW \times (1 - Dram\_HitRate) \quad (17)$$

$$W = App\_WrBW \quad (18)$$

$$WB = Dram\_Dirty \times ((1 - Dram\_HitRate) \times (App\_RdBW + App\_WrBW)) \quad (19)$$

Eq. (19) defines the bandwidth that is used to write back dirty cachelines to NVM on DRAM read misses and write misses. In both cases, the dirty cacheline is replaced by another line. A read hit and a write hit do not trigger a writeback as no

Table 5. Delivered read and write bandwidths in DRAM+NVM heterogeneous memory systems with DRAM cache

$DRAM\_RdBW\_Delivered = \min\left(DRAM\_RdBW\_Demand, \frac{DRAM\_RdBW}{1+\frac{DRAM\_WrBW\_Demand}{DRAM\_RdBW\_Demand}}\right)$	(23)
$DRAM\_WrBW\_Delivered = \min\left(DRAM\_WrBW\_Demand, \frac{DRAM\_RdBW}{1+\frac{DRAM\_RdBW\_Demand}{DRAM\_WrBW\_Demand}}\right)$	(24)
$NVM\_RdBW\_Delivered = \min\left(\frac{RM}{S_{DRAM}}, \frac{NVM\_RdBW}{1+\frac{NVM\_R2W}{(RM/WB)}}\right), S_{DRAM} = \frac{DRAM\_RdBW\_Demand}{DRAM\_RdBW\_Delivered}$	(25)
$NVM\_WrBW\_Delivered = \min\left(\frac{WB}{S_{DRAM}}, \frac{NVM\_RdBW}{NVM\_R2W+\frac{RM}{WB}}\right), S_{DRAM} = \frac{DRAM\_RdBW\_Demand}{DRAM\_RdBW\_Delivered}$	(26)

cacheline is replaced. Note that a clean cacheline is never written back to NVM.

We can now express bandwidth demand on the DRAM cache using *RH*, *RM*, *W*, and *WB*.

$$DRAM\_RdBW\_Demand = RH + WB \quad (20)$$

$$DRAM\_WrBW\_Demand = RM + W \quad (21)$$

The bandwidth demand on the DRAM cache when NVM bandwidth is not a constraint (*DRAM_RdBW_Demand* and *DRAM_WrBW_Demand*) can replace application bandwidth demand in Equations (2) and (3). Thus, we derive the bandwidth delivered by the DRAM cache (*DRAM_RdBW_Delivered* and *DRAM_WrBW_Delivered*) when NVM bandwidth is not a limiting factor using Equations (23) and (24) in Table 5.

Using bandwidth demand on the DRAM cache and bandwidth delivered by the DRAM cache, we can find out to what degree bandwidth is throttled by DRAM. We define a term called the scale-down factor due to the DRAM cache bandwidth (S_{DRAM}) to account for this bandwidth reduction. S_{DRAM} is calculated as follows:

$$S_{DRAM} = \frac{DRAM\_RdBW\_Demand}{DRAM\_RdBW\_Delivered} \quad (22)$$

Next, we find out the bandwidth delivered by NVM (*NVM_RdBW_Delivered* and *NVM_WrBW_Delivered*). *RM* dictates read bandwidth demand on NVM and *WB* dictates write bandwidth demand on NVM. *RM* and *WB* can replace application bandwidth demand in Equations (2) and (3). Thus, we derive the bandwidth delivered by NVM using Equations (25) and (26) in Table 5.

Next, we need to find out whether the bandwidth delivered by NVM can keep up with the bandwidth delivered by the DRAM cache (i.e., is $(RM / S_{DRAM}) < NVM\_RdBW\_Delivered$ and $(WB / S_{DRAM}) < NVM\_WrBW\_Delivered$?). If the DRAM cache consumes bandwidth at a rate at which NVM cannot keep up with, then the DRAM cache bandwidth is throttled by the NVM bandwidth. We define a term called the scale-down factor due to NVM bandwidth (S_{NVM}) to account for this bandwidth reduction. S_{NVM} is calculated as following:

$$S_{NVM} = \frac{RM/S_{DRAM}}{NVM\_RdBW\_Delivered} \quad (27)$$

Using S_{DRAM} and S_{NVM}, we can scale down bandwidth numbers for *RM*, *RH*, *WB*, and *W* to derive the scaled variants of these numbers. For example, the scaled variant of *RM* is calculated as $RM / (S_{DRAM} \times S_{NVM})$.

In brief, we can calculate the bandwidth delivered by DRAM cache and NVM by following the steps below.

1. Calculate bandwidth delivered by DRAM cache (*DRAM_RdBW_Delivered* and *DRAM_WrBW_Delivered*) when NVM bandwidth is not a limiting factor using Equations (23) and (24).
2. Calculate bandwidth delivered by NVM (*NVM_RdBW_Delivered* and *NVM_WrBW_Delivered*) using Equations (25) and (26).
3. Adjust bandwidth numbers (*RM*, *RH*, *WB*, and *W*) using scale-down factors (S_{DRAM} and S_{NVM}).

Following the steps above, we calculate memory bandwidth in a hardware-managed-cache-based heterogeneous memory system with SRAM tag storage.

3.2.1 Read to Write Bandwidth Ratio

We analyze read and write bandwidth demand on NVM and DRAM to gain insight into the relationship between bandwidth requirements and DRAM hit rate. The read and write bandwidth demand on NVM is defined by *RM* and *WB*, which are functions of DRAM cache hit rate and application read and write bandwidth demands. NVM read bandwidth demand has a linear-inverse relationship with the DRAM hit rate as shown in Figure 6a (see Eq. (17) for reference).

NVM write bandwidth demand exhibits a more complex behavior as it is affected by multiple factors including the probability of cacheline dirtiness and both application read and write bandwidths. By substituting the expression for

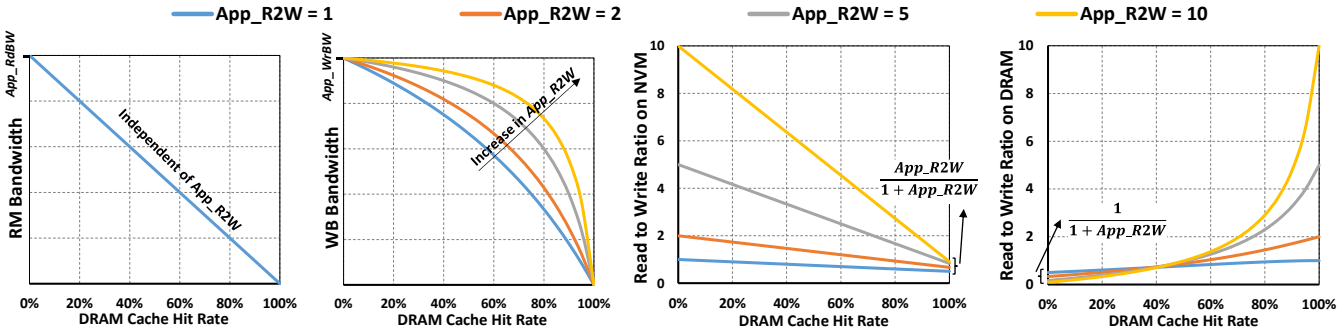

Figure 6. Bandwidth demand on NVM and DRAM cache as a function of the DRAM cache hit rate.

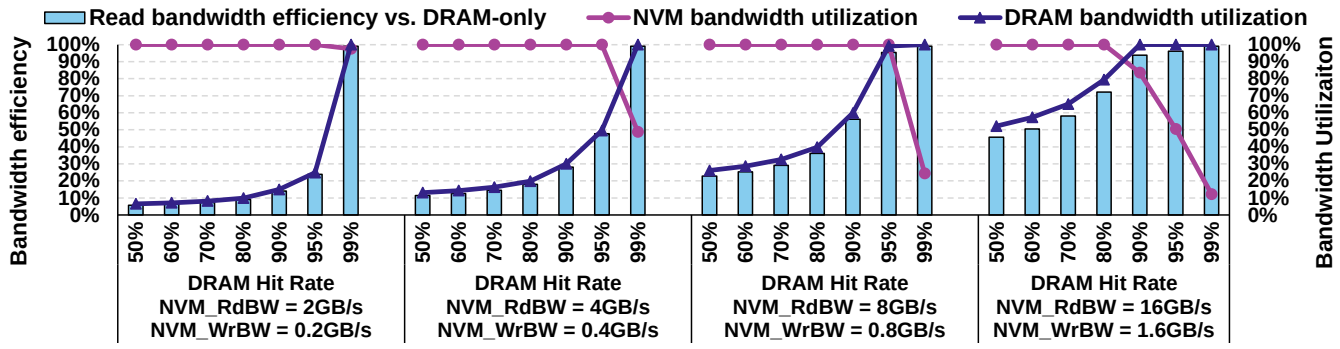

Figure 7. Bandwidth efficiency and utilization in DRAM+NVM heterogeneous memory systems with DRAM cache with SRAM tag storage (DRAM_RdBW = 19GB/s, App_R2W = 5).

Dram_Drity from Eq. (6) into Eq. (19), we rewrite the expression for *WB*.

$$WB = \frac{(1-Dram\_HitRate)\times(App\_RdBW + App\_WrBW)}{1+App\_R2W\times(1-Dram\_HitRate)} \quad (28)$$

Figure 6b shows the relationship between NVM write bandwidth, DRAM hit rate, and application read and write bandwidth. *WB* is not only dependent on *App_WrBW*, but it is also dependent on *App_RdBW*.

Using *RM* and *WB* expressions, we calculate the ratio of read to write bandwidth demand on NVM as shown in Figure 6c.

$$\frac{RM}{WB} = \frac{App\_RdBW(App\_WrBW+App\_RdBW\times(1-Dram\_HitRate))}{App\_WrBW(App\_RdBW+App\_WrBW)} \frac{App\_R2W(1+App\_R2W\times(1-Dram\_HitRate))}{1+App\_R2W} \quad (29)$$

At the two opposite extremes, when DRAM hit rate is 0% and approaching 100%, *RM / WB* evaluates to *App_R2W* and *App_R2W / (1 + App_R2W)*, respectively. Figure 6c reveals that the ratio of read to write bandwidth demand on NVM is linearly dependent on DRAM miss rate and the ratio decreases as the DRAM hit rate increases.

Similarly, we can calculate the ratio of read to write bandwidth demand on the DRAM cache (*DRAM_RdBW_Demand / DRAM_WrBW_Demand*) which is shown in Figure 6d. At the two opposite extremes, when DRAM hit rate is 0% and 100%, *DRAM_RdBW_Demand / DRAM_WrBW_Demand* evaluates to *1 / (1 + App_R2W)* and *App_R2W*, respectively. One important observation is that, as DRAM hit rate increases, the ratio of read to write bandwidth demand on DRAM cache increases while the ratio of read to write bandwidth demand on NVM decreases linearly.

3.2.2 Bandwidth Efficiency and Utilization

The bars in Figure 7 show the total read bandwidth from the application's perspective achieved in this memory system relative to a DRAM-only system. In other words, the bars show the read bandwidth efficiency achieved by different DRAM cache and NVM configurations relative to a DRAM-only system. The total read bandwidth achieved is the sum of the read hit bandwidth (*RH*) and read miss bandwidth (*RM*). In this figure, we assume that *DRAM_RdBW* is 19GB/s and the application is memory bandwidth bound with *App_R2W* of 5. For this configuration, the maximum achievable read bandwidth in a DRAM-only system is 15.83GB/s (5/6 × 19GB/s). We vary the DRAM cache hit rate and NVM bandwidth to explore the bandwidth efficiency of different configurations. The lines in Figure 7 show the total (read and write) utilizations of DRAM and NVM bandwidths, including bandwidth overheads.

There are some key observations from Figure 7. First, DRAM cache hit rate has a huge impact on the achievable read bandwidth. For example, for an NVM with a 2GB/s read and 0.2GB/s write bandwidth, only 10% of the maximum read bandwidth is achievable when the DRAM cache hit rate is

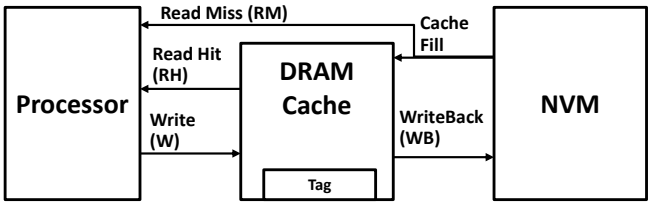

Figure 8. Logical view of the intelligent DRAM cache.

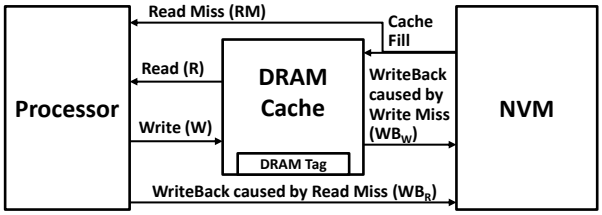

Figure 9. Logical view of the write-intelligent DRAM cache with DRAM tag storage.

80%. When the DRAM cache hit rate gets close to 99%, the achievable read bandwidth jumps up to 99%. Second, when the DRAM cache hit rate is relatively low (<95%), the NVM bandwidth affects the bandwidth efficiently significantly, while NVM bandwidth is not a major factor for systems with high DRAM cache hit rates. Third, NVM bandwidth and DRAM cache bandwidth affect each other. For some configurations, NVM bandwidth is fully utilized while DRAM bandwidth is underutilized (e.g., when the hit rate is low and NVM bandwidth is low). For other configurations, DRAM bandwidth is fully utilized while NVM bandwidth is underutilized (e.g., when hit rate is very high). Fourth, increasing NVM bandwidth increases DRAM bandwidth utilization, which illustrates that DRAM bandwidth was underutilized because NVM bandwidth could not keep up with bandwidth demand by DRAM.

3.3 Intelligent DRAM Cache

In our next study, we assume the DRAM cache incorporates logic that handles read and write requests without the processor's help and tags for the DRAM cache are placed in the DRAM package (Figure 8). Tags can be implemented using either DRAM cells or SRAM cells, the latter of which may be in a logic die within the DRAM package. In this organization, the DRAM cache is able to fetch tags internally, perform tag comparison, handle data replacement, and request data from NVM. Such an organization may be desirable in future 3D-stacked DRAM packages where a logic die within the package can provide cache control functions. We refer to such a DRAM cache as an intelligent DRAM cache.

We start by studying the DRAM bandwidth for reads and writes. For reading cachelines, the processor sends a request to DRAM and waits for a response. The intelligent DRAM cache performs tag comparison to determine a hit or miss. In the case of a cache hit, the intelligent DRAM responds with data. In the case of a miss, the intelligent DRAM initiates miss handling, performs a cacheline replacement, and then responds with the requested data. If the existing cacheline is clean, the intelligent DRAM fetches the requested data from NVM. If the existing cacheline is dirty, the intelligent DRAM cache evicts (writes back) the line from the cache and fetches the requested data from NVM. This process results in variable data read latency from the processor's point of view, depending on cacheline residence, dirtiness of data, and NVM bandwidth utilization.

For writing cachelines, the processor sends data to DRAM and the intelligent DRAM cache performs tag comparison. In the case of a cache hit, the intelligent DRAM cache replaces the data. In the case of a miss, if the existing data is dirty, the intelligent DRAM cache evicts the line to NVM which consumes DRAM read bandwidth and NVM write bandwidth. Next, the new tag and data are written to DRAM cache.

In the case where tags are stored in DRAM, we assume that the intelligent DRAM cache has greater internal bandwidth for fetching DRAM tags (e.g., by storing tags in dedicated internal banks or using 3D-stacked DRAMs with higher internal bandwidth). Thus, tag fetch does not consume data bandwidth that would otherwise be utilized by the processor and does not block the processor's access to data. In Section 3.5, we provide a DRAM cache model where the tag fetch overhead affects the processor's DRAM data bandwidth. Considering these assumptions, the bandwidth expressions for the variables *RH*, *RM*, *W*, and *WB* in this model are the same as those in the DRAM cache with tag storage in the processor. As a result, greater internal bandwidth and intelligence in DRAM negate the tag fetch overhead for both reads and writes. Since the intelligent DRAM cache provides the same bandwidth efficiency and utilization as the DRAM cache with SRAM-tag, we do not present any analytical results for it.

3.4 Write-intelligent DRAM Cache

In our next study, we assume tags for the DRAM cache are stored in DRAM and the DRAM cache features intelligent logic that handles only write requests without the processor's help. Figure 9 shows a logical view of the write-intelligent DRAM cache. As discussed in Section 3.3, using greater internal bandwidth and intelligent logic in the DRAM cache for tag fetch results in reduced external DRAM bandwidth demand. In this organization, the processor performs tag fetch and comparison for read requests, but hands over processing write requests to the DRAM intelligent logic. Since writes are off the critical path, the intelligent logic in the DRAM cache opportunistically uses internal bandwidth for fetching tags for write requests, which lowers external bandwidth demand.

In order to estimate the impact of tag reads on the processor-DRAM interface bandwidth, we adopt the Alloy DRAM cache model [17] where the DRAM is a direct-mapped, non-inclusive cache that uses a write-allocate policy and a non-power-of-two number of sets.[1] In Alloy DRAM cache, cache

[1] This entails storing entire address (excluding cacheline offset) in DRAM.

data lines are 64 bytes long and tags and miscellaneous bits are 16 bytes long. Each time the processor accesses the DRAM cache, it transfers 80 bytes of tag and data that are placed side by side in DRAM. While we base our analysis on the Alloy cache, our models extend to other variants as well.

We start by studying the DRAM bandwidth for reads and writes. For reads, the processor needs to fetch both data and tag from DRAM to determine cache hit or miss. We assume that both the tag and data are fetched with a single memory request to reduce memory access latency. It is possible to read tags first and then data on a tag match (to save bandwidth), but this results in increased read hit latency. Thus, we assume the processor fetches both tag and data (80 bytes) with a single request, taking a burst of five DRAM transfers [17]. If the tag matches (cache hit), 20% (16 bytes out of 80 bytes) of DRAM bandwidth is "wasted" (i.e., not useful from the application's perspective). If the tag does not match (cache miss) and the data is dirty, similarly 20% of DRAM bandwidth is wasted as dirty data is used by the processor for writing back data to NVM. That is why there exist two *logical* writeback bandwidths in Figure 9 (Note that, as shown in Figure 2, an actual physical implementation only has one write datapath to NVM). However, if the fetched data on a miss is clean, the entire DRAM bandwidth is wasted. On a read miss, the processor initiates a DRAM cache fill from NVM which consumes DRAM write bandwidth. The intelligent logic in the DRAM cache handles the tag update. In brief, for every read request, 80 bytes are fetched from DRAM, resulting in a 1.25× increase in DRAM bandwidth.

For writes, the processor sends data to DRAM and the intelligent cache management logic performs tag comparison to indicate a cache hit or miss and handles the write request accordingly. In the case of a cache hit, the intelligent DRAM cache replaces the data. In the case of a miss, if the existing data is dirty, the intelligent DRAM cache evicts the line to NVM which consumes DRAM read bandwidth. Then, the new data and tag are written to the DRAM cache.

We can now write the bandwidth expressions for the variables *R*, *RM*, *W*, and *WB* in this model. We logically break writeback bandwidth (*WB*) into two separate expressions. WB_W is the writeback bandwidth of dirty lines replaced by write misses. WB_R is the writeback bandwidth of dirty lines replaced by read misses.

$$R = 1.25 \times App\_RdBW \quad (30)$$

$$RM = App\_RdBW \times (1 - Dram\_HitRate) \quad (31)$$

$$W = App\_WrBW \quad (32)$$

$$WB_W = Dram\_Dirty \times (1 - Dram\_HitRate) \times App\_WrBW \quad (33)$$

$$WB_R = Dram\_Dirty \times (1 - Dram\_HitRate) \times App\_RdBW \quad (34)$$

As can be seen from Equations (30)-(34), only the *R* expression here is different from a DRAM cache with SRAM tags. Due to the latency reduction policy of fetching the combined tag and data, $1/1.25 \times Dram\_HitRate = 0.8 \times Dram\_HitRate$ fraction of the *R* bandwidth is useful from the application's perspective and the rest is wasted on fetching tags and missed data. For instance, with a DRAM hit rate of 60%, only 48% of the total *R* bandwidth is useful. It is worth mentioning that although the dirty, missed cachelines fetched by the processor are used for writing to NVM (WB_R), they are not considered useful data to the application.

Bandwidth demand on the DRAM is expressed as:

$$DRAM\_RdBW\_Demand = R + WB_W \quad (35)$$

$$DRAM\_WrBW\_Demand = RM + W \quad (36)$$

As for bandwidth demand on NVM, similar to Section 3.2, *RM* dictates the read bandwidth demand on NVM and *WB* ($WB_R + WB_W$) dictates the write bandwidth demand on NVM.

With the updated *R* expression, we can calculate the bandwidth delivered by DRAM cache and NVM by following the steps presented in Section 3.2.

3.4.1 Read to Write Bandwidth Ratio

The ratio of read to write bandwidth on NVM in a write-intelligent DRAM cache is the same as the ratio of read to write bandwidth on NVM in a DRAM cache with SRAM tags (Figure 6c). The reason is that the *RM* and *WB* bandwidth expressions in both memory systems are the same.

However, the read bandwidth to write bandwidth ratio at the write-intelligent DRAM cache (*DRAM_RdBW_Demand* / *DRAM_WrBW_Demand*) is different from that of a DRAM cache with SRAM tags and evaluates to:

$$\frac{1.25 \times App\_R2W + \frac{1 - Dram\_HitRate}{1 + App\_R2W \times (1 - Dram\_HitRate)}}{App\_R2W \times (1 - Dram\_HitRate) + 1} \quad (37)$$

When DRAM hit rate approaches 100%, this ratio equals $1.25 \times App\_R2W$. Compared to DRAM cache with SRAM-tag, the write-intelligent DRAM cache has higher read to write bandwidth demand for the same values of *App_R2W* and *Dram_HitRate*.

3.4.2 Bandwidth Efficiency and Utilization

Figure 10 shows the bandwidth efficiency and utilization in this heterogeneous memory system. We make the same assumptions as in Section 3.2 and we similarly vary DRAM cache hit rate and NVM read and write bandwidth for design exploration purposes. Similar to a DRAM cache with SRAM-tag storage, DRAM hit rate and NVM bandwidth have significant impact on bandwidth efficiency. However, the bandwidth efficiency of this system is lower than that of SRAM tag storage when the DRAM cache hit rate is high. For example, the maximum efficiency reported in Figure 10 is 82%. This reduction is due to the tag fetch overhead in this memory system.

In addition, DRAM bandwidth utilization of this memory system is equal to or higher than that of SRAM tag storage as more DRAM bandwidth is wasted on unused data and tag overhead. In contrast, NVM bandwidth utilization of this memory system is equal to or lower than that of SRAM-tag storage. When DRAM bandwidth utilization is 100%, NVM bandwidth could be underutilized. As some DRAM bandwidth is wasted in this memory system, the NVM bandwidth

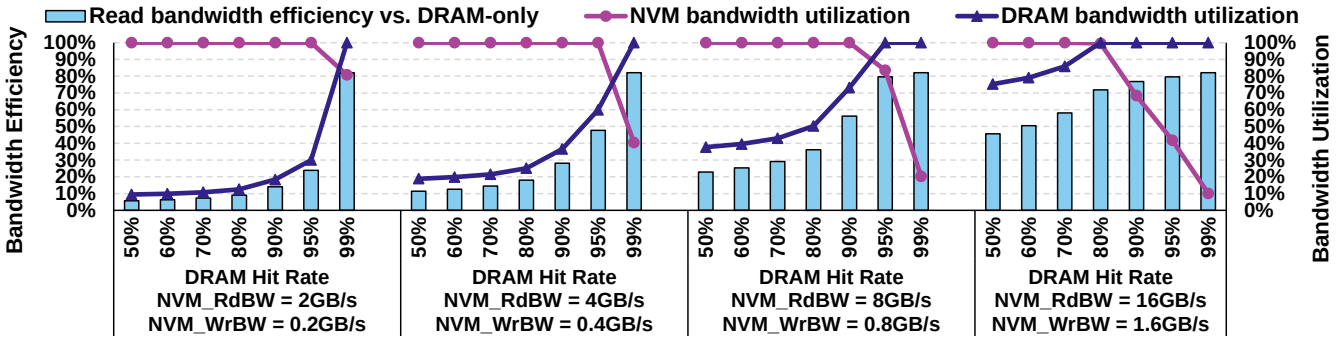

Figure 10. Bandwidth efficiency and utilization in DRAM+NVM heterogeneous memory systems with write-intelligent DRAM cache with DRAM tag storage (DRAM_RdBW = 19GB/s, App_R2W = 5).

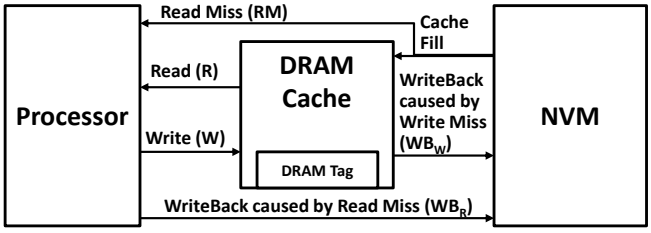

Figure 11. Logical view of the unintelligent DRAM cache with DRAM tag storage.

is more severely underutilized in these cases compared to SRAM tag storage.

3.5 Unintelligent DRAM Cache

In our next study, we evaluate bandwidth of an "unintelligent" DRAM-cache based heterogeneous memory system with DRAM tags. We make the same assumptions as those explained in Section 3.4, but the DRAM cache is implemented as a simple DRAM module with no intelligence for tag comparison and cacheline replacement. All tag comparisons and cache management decisions are performed by the processor. A logical view of such an organization is shown in Figure 11. Similar to the write-intelligent DRAM cache, the physical structure (Figure 2) of this organization requires only one write datapath to NVM.

Reads in this organization are handled in the same way as a write-intelligent DRAM cache. The only difference is that in the case of a DRAM cache read miss, the processor should also write the tag to the DRAM cache as the DRAM cache does not feature intelligent logic to compute tags.

However, writes are handled differently. For writes, the processor first needs to fetch the tag to determine whether the access is a hit or miss in the DRAM cache. This results in consuming additional DRAM read bandwidth.[2] In the case of a hit, the new data is written. In the case of a miss, if the existing data is dirty, the processor fetches it from the DRAM cache and writes it to NVM (WB_W) which consumes DRAM read bandwidth.[3] The processor then writes the new tag and data. In either case, 20% of DRAM bandwidth is wasted for writing tags. In brief, this organization incurs more bandwidth overhead for fetching and updating tags compared to previously discussed designs.

We can now rewrite the expressions for the variables *R*, *RM*, *W*, and *WB* in this model.

$$R = 1.25 \times App\_RdBW + 0.25 \times App\_WrBW \quad (38)$$

$$RM = App\_RdBW \times (1 - Dram\_HitRate) \quad (39)$$

$$W = App\_WrBW + 0.25 \times App\_WrBW \times (1-Dram\_HitRate) + 0.25 \times App\_RdBW \times (1-Dram\_HitRate) \quad (40)$$

$$WB_W = Dram\_Dirty \times (1 - Dram\_HitRate) \times App\_WrBW \quad (41)$$

$$WB_R = Dram\_Dirty \times (1 - Dram\_HitRate) \times App\_RdBW \quad (42)$$

For the *W* expression, we assume that existing metadata are not updated for read and write hits. Otherwise, even more write bandwidth overheads will be incurred. Note that not all of the *R* and *W* bandwidths is useful and a fraction is wasted on tag fetching and missed data fetching. Bandwidth demand on the DRAM is expressed as:

$$DRAM\_RdBW\_Demand = R + WB_W \quad (43)$$

$$DRAM\_WrBW\_Demand = RM + W \quad (44)$$

With these updated expressions, we can similarly calculate the bandwidth delivered by the DRAM cache and NVM by following the steps presented in Section 3.2.

3.5.1 Read to Write Bandwidth Ratio

The ratio of read to write bandwidth on NVM in this memory system is the same as in the DRAM cache with SRAM-tag (Figure 6c) as *RM* and *WB* bandwidth expressions are the same. However, the read bandwidth to write bandwidth ratio on the DRAM cache (*DRAM_RdBW_Demand* / *DRAM_WrBW_Demand*) is different and is derived as:

$$\frac{1.25\times App\_R2W+0.25+\frac{1-Dram\_HitRate}{1+App\_R2W\times(1-Dram\_HitRate)}}{1.25\times App\_R2W\times(1-Dram\_HitRate)+0.25\times(1-Dram\_HitRate)+1} \quad (45)$$

[2] There are proposals to avoid fetching tags from DRAM (e.g., the DRAM Cache Presence technique [7]) but require changing the processor's LLC.

[3] We assume the processor fetches the tag and existing dirty data serially in the case of write miss. Alternatively, the processor can speculatively fetch data and tag in parallel. This reduces latency, but wastes bandwidth if existing data is not dirty.

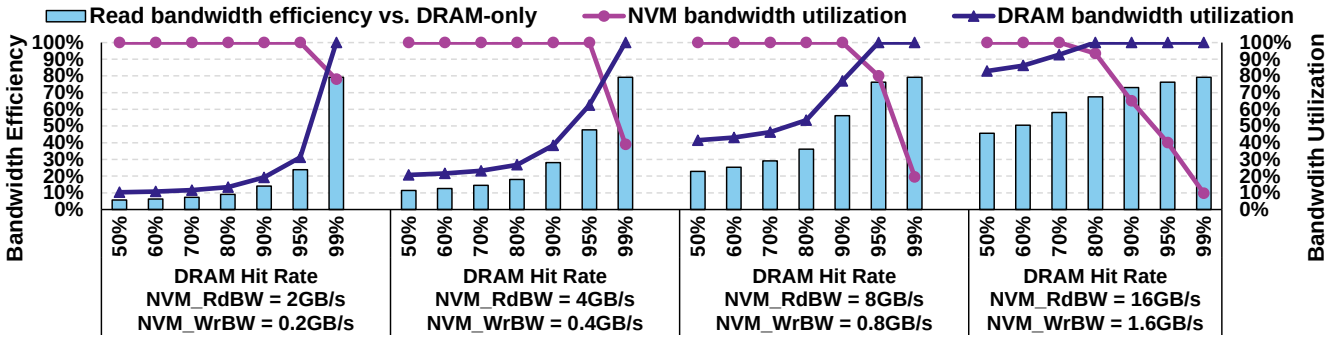

Figure 12. Bandwidth efficiency and utilization in DRAM+NVM heterogeneous memory systems with unintelligent DRAM cache with DRAM tag storage (DRAM_RdBW = 19GB/s, App_R2W = 5).

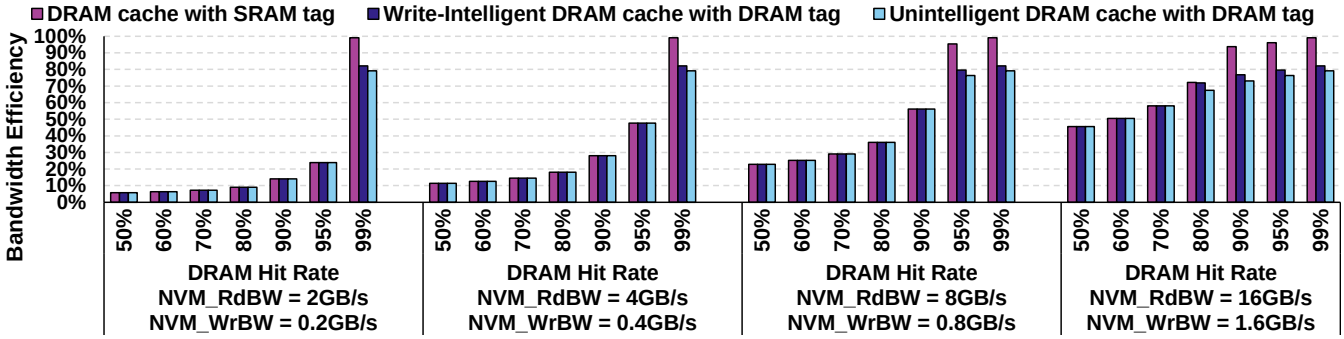

Figure 13. Bandwidth efficiency of different DRAM+NVM heterogeneous memory systems with varying NVM bandwidth (DRAM_RdBW = 19GB/s, App_R2W = 5). NVM read to write bandwidth ratio is 10 across all bars.

When the DRAM hit rate approaches 100%, this ratio equals $1.25 \times App\_R2W + 0.25$.

3.5.2 *Bandwidth Efficiency and Utilization*

Figure 12 shows the bandwidth efficiency and utilization in this heterogeneous memory system. We make the same assumptions as in Section 3.2 and we similarly vary DRAM cache hit rate and NVM read and write bandwidth for design exploration purposes. Similar to previous systems studied, DRAM hit rate and NVM bandwidth have significant impact on bandwidth efficiency. However, bandwidth efficiency of this system is even lower than the two previous systems when DRAM cache hit rate is high (with a maximum of 79%). The reason is higher tag fetch and update overhead in this memory system relative to the previously discussed organizations.

4. Discussion of Bandwidth Efficiency

In this section, we compare the bandwidth efficiency of the three cache-based heterogeneous memory systems described in Section 3 under different configurations. As defined earlier, the bandwidth efficiency of DRAM-cache-based systems indicates the total read bandwidth achieved from the application's perspective relative to the read bandwidth of a DRAM-only system. In our models, the DRAM cache with SRAM tags and intelligent DRAM cache feature the same bandwidth efficiency. Hence, we do not present results for the intelligent DRAM cache.

Figure 13 compares the bandwidth efficiency of the three DRAM-cache-based memory systems under different NVM bandwidths. We make the same assumptions as in Section 3.2 and similarly vary DRAM cache hit rate and NVM read and write bandwidth for design exploration purposes. The NVM read to write bandwidth ratio is kept constant. When the DRAM hit rate is low, these systems achieve the same bandwidth efficiency because at low hit rates, a large number of requests are fulfilled by NVM, causing NVM bandwidth to become the bottleneck. As a result, DRAM bandwidth remains underutilized, negatively impacting the system bandwidth efficiency. This underutilization means that DRAM can fulfill bandwidth overheads for tags and data misses with no impact on overall efficiency. For instance, when NVM read bandwidth is 2GB/s, even at a hit rate of 95%, these systems have the same bandwidth efficiency (24%). When NVM read bandwidth increases to 8GB/s, differences in their bandwidth efficiency are more pronounced (76-95% at a hit rate of 95%). This suggests that when NVM bandwidth is much lower than DRAM bandwidth, the organization of the tags and memory system does not affect how much bandwidth the memory system can deliver simply because the bottleneck is shifted to NVM bandwidth. So when NVM bandwidth is very low, one could use the most cost-effective or simplest memory system without impacting performance.

However, when NVM bandwidth is higher (e.g., NVM read bandwidth of 16GB/s), the more efficient DRAM cache implementations (i.e., with SRAM tags or intelligent in-DRAM tag management for read and writes) achieves better bandwidth efficiency at high hit rates. Another observation is that

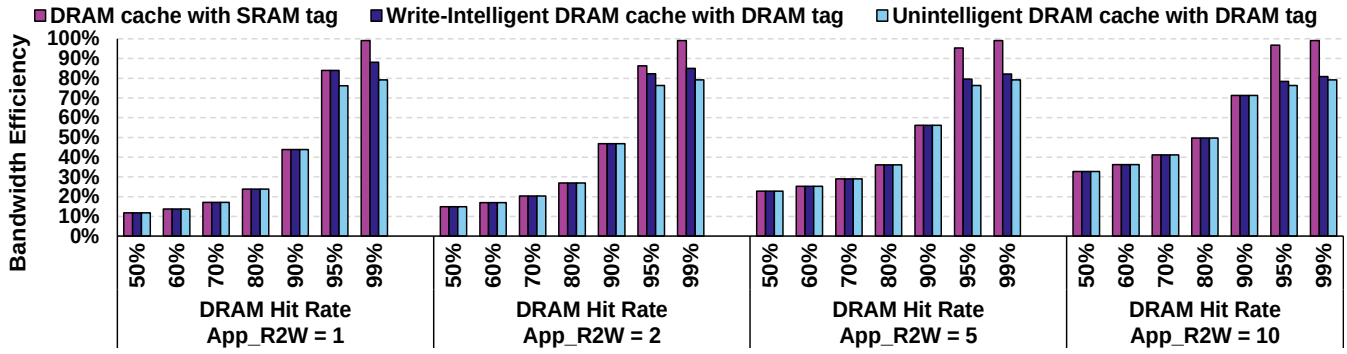

Figure 14. Bandwidth efficiency of different DRAM+NVM heterogeneous memory systems with varying application read to write bandwidth ratio (DRAM_RdBW = 19GB/s, NVM_RdBW = 8GB/s, NVM_WrBW = 0.8GB/s).

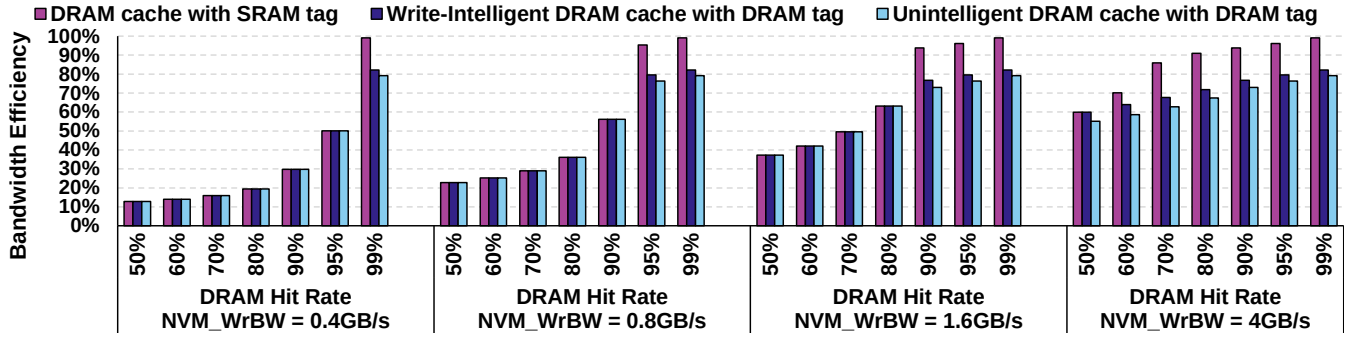

Figure 15. Bandwidth efficiency of different DRAM+NVM heterogeneous memory systems with varying NVM write bandwidth (DRAM_RdBW = 19GB/s, NVM_RdBW = 8GB/s, App_R2W = 5).

a write-intelligent DRAM cache is only marginally better than an unintelligent DRAM in terms of bandwidth efficiency even when the DRAM hit rate is high.

Figure 14 compares the bandwidth efficiency of the three cache-based heterogeneous memory systems under varying application read to write ratios. In this figure, we assume that NVM read bandwidth and write bandwidth are 8GB/s and 0.8GB/s, respectively. Also, DRAM bandwidth is set to 19GB/s. When the DRAM hit rate is very high (e.g., 99%), the application read to write ratio has a very limited impact on bandwidth efficiency as most of the bandwidth is filtered by DRAM, which has symmetric read and write bandwidths. However, when DRAM hit rate is low (e.g., 60%), as the application read to write ratio increases, bandwidth efficiency increases. The reason for the increase in bandwidth efficiency is that NVM write bandwidth is much lower than NVM read bandwidth, thus negatively impacting bandwidth efficiency more when most of the memory requests that cannot be filtered by the DRAM cache are writes. The limited NVM bandwidth (especially, write bandwidth) also reduces DRAM bandwidth utilization. As the application read to write ratio increases, the negative impact of limited NVM write bandwidth on DRAM bandwidth utilization diminishes, which results in increased DRAM bandwidth utilization.

Another observation from Figure 14 is that, for cases with very high DRAM cache hit rates, the difference in bandwidth efficiency of the write-intelligent DRAM system and that of the unintelligent DRAM system shrinks as the application read to write ratio increases because the write-intelligent DRAM is beneficial only when write requests are abundant. Note that when the DRAM hit rate is low, the difference in bandwidth efficiency between these two systems is discernible only when DRAM bandwidth utilization approaches 100%.

Figure 15 shows the impact of NVM write bandwidth on the bandwidth efficiency of different heterogeneous memory systems. In this figure, we hold NVM read bandwidth and DRAM bandwidth constant at 8GB/s and 19GB/s, respectively. The application read to write bandwidth ratio is also held constant at 5. NVM write bandwidth is varied from 0.4GB/s to 4GB/s (5% to 50% of NVM read bandwidth). This figure shows that NVM write bandwidth has a significant impact on bandwidth efficiency even though bandwidth efficiency measures the read bandwidth from an application's perspective. The reason is that the total NVM bandwidth is shared between reads and writes with writes being more time-consuming. By increasing the NVM write bandwidth, DRAM utilization (and thus bandwidth efficiency) improves as NVM can fulfill more requests. On the other hand, when the DRAM hit rate is very high (e.g., 99%), NVM write bandwidth is not a major limiting factor as most requests are satisfied by DRAM without requiring NVM accesses.

5. Latency Breakdown

In this section, we analyze the latency of different heterogeneous memory organizations. Table 6 summarizes the latency incurred to service requests from DRAM (hit in the DRAM

Table 6. Latency breakdown

Flat-address NUMA	
DRAM:	DRAM data read + DRAM bus
NVM:	NVM data read + NVM bus
DRAM cache with SRAM tag	
Hit:	SRAM tag read + DRAM data read + DRAM bus
Miss:	SRAM tag read + NVM data read + NVM bus
Intelligent DRAM cache	
Hit:	DRAM tag read + DRAM data read + DRAM bus
Miss:	DRAM tag read + NVM data read + NVM bus
Write-intelligent and unintelligent DRAM caches (Both use Alloy cache model)	
Hit:	combined DRAM tag and data read + DRAM bus
Miss:	combined DRAM tag and data read + DRAM bus + NVM data read + NVM bus

cache) and NVM (miss in the DRAM cache). We do not include the processor's cache hierarchy latency and memory queuing latency as those would be relatively unchanged across different memory organizations. In Table 6, "read" indicates the latency to read from memory cells and "bus" indicates the latency to transfer data over the off-chip interface to the processor.

To have a better sense of the latency variation across different organizations, we visualize the latency expressions in Figure 16. We consider the latency of a single access to the heterogeneous memory in the absence and presence of row buffer locality (RBL). When there is RBL, memory requests hit in the row buffer and are serviced with a latency of column access time (tCL). When there is no RBL, the memory requests miss in the row buffer and are serviced with a latency of row cycle time (tRC). Note that tRC=tRAS+tRP and tRP is typically 0 in NVM. In either case, data is transferred over the off-chip bus interface with a latency of tBURST. We assume that DRAM and NVM have a tCL of 14ns, DRAM has a tRC of 49ns and tBURST of 5ns, NVM has a tRC of 69ns and tBURST of 10ns, and SRAM tag read takes 2ns.

Figure 16 shows that latency incurred could be very broad depending on row buffer locality and the number of required accesses. For instance, when there is a miss in DRAM cache, additional latency is incurred to access NVM. In the intelligent DRAM cache, two separate memory accesses are always required since tag and data are stored in separate locations. This increases latency significantly especially when both accesses suffer from row buffer conflicts. In write-intelligent and unintelligent DRAM caches, however, tag and data are stored sequentially in accordance with the Alloy cache model (Section 3.4). Thus, DRAM cache hits require only one memory access.

As expected the flat-address NUMA organization incurs the lowest latency since there is not tag check overhead. DRAM cache with SRAM tags has slightly higher latency due to SRAM tag lookup. Other DRAM caches have higher latency especially when tag and data fetch are not combined in a single access.

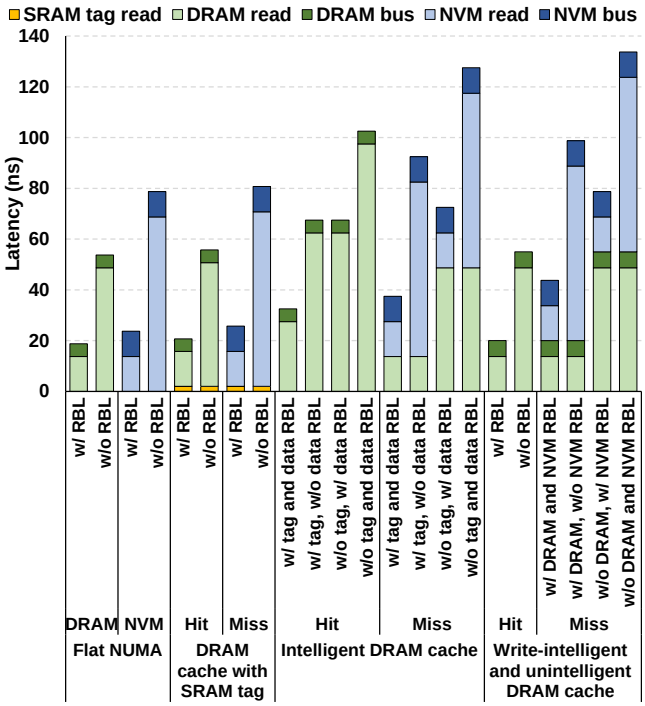

Figure 16. Latency breakdown for different organizations. RBL stands for row buffer locality. DRAM and NVM accesses are shown in green and blue, respectively.

6. Related Work

Prior work on heterogeneous memory systems has proposed using DRAM as either an OS-managed flat memory or an extra level of cache. When used as a cache, tags can be stored in a separate SRAM structure or in DRAM. Storing tags in SRAM incurs high storage overhead for large caches, which makes it impractical for high-capacity DRAM caches. Storing tags in DRAM, on the other hand, has its own challenges including bandwidth overheads for accessing tags from the processor or complexity of incorporating tag management logic within the DRAM package.

Prior proposals for heterogeneous memory organizations do not perform analytical studies for bandwidth estimation. The closest work to ours that tries to analytically model heterogeneous memory systems is done by Bolotin *et al.* [4]. However, they disregard bandwidth asymmetry and different organizations of DRAM-cache-based systems.

DRAM-cache-based memory systems. The DRAM cache proposal by Loh and Hill [13, 14] uses a 29-way set associative cache and co-locates tags and data for all ways of a set in the same DRAM row. The Loh-Hill cache is optimized for hit rate, as each cache access requires fetching all tags first and then the matching line, incurring high bandwidth overhead. To quickly indicate a hit or miss, they use a MissMap SRAM structure alongside the LLC. Alloy cache [17] is a latency-optimized DRAM cache organization that fetches the tag and data from DRAM together in a single burst to eliminate tag and data serialization. Alloy cache is more bandwidth efficient than the Loh-Hill cache as only 1.25 lines are fetched

per request. Hence, our bandwidth models for DRAM caches with DRAM-tag are based on the Alloy organization. Further, our models use a direct-mapped DRAM cache as in Alloy and Intel's Knights Landing [20]. The BEAR DRAM cache [7] entails three techniques to further optimize bandwidth efficiency in DRAM caches with DRAM-tag. Using a relatively small amount of SRAM storage, BEAR selectively bypasses cache fills on misses, selectively eliminates fetching tags from DRAM for writebacks from LLC to DRAM, and reduces the bandwidth overhead of fetching tags on reads. Unlike previous caches, the Footprint DRAM cache [10] is a page-based cache in which a single SRAM tag per page is used to reduce tag storage overhead. Only predicted blocks of a page are transferred to reduce wasted off-chip bandwidth. Similarly, Unison DRAM cache [9] is a page-based, set-associative cache. But tags and data for pages in a set are co-located in the same DRAM row. Unison uses a way predictor to fetch a single tag and a data block in a back-to-back transfer.

Our models can be extended to cover other organizations of block-based and page-based heterogeneous memory systems [13, 14, 10, 9, 12], hit-miss and line predictors [13, 10, 9, 6], and bandwidth optimization techniques [7].

7. Conclusions

We analytically studied the bandwidth efficiency and bandwidth utilization of a range of heterogeneous memory systems. We separate read and write bandwidths and consider the impact of low write bandwidth of NVM on the bandwidth efficiency of heterogeneous memory systems. Below is a summary of our findings.

In NUMA flat-address memory systems:

- The maximum bandwidth efficiency (i.e., total delivered read bandwidth) is achieved when both DRAM and NVM bandwidths are fully utilized (100% utilization).
- Data placement between DRAM and NVM has direct impact on bandwidth efficiency. The optimal data placement for bandwidth maximization is a function of not only the bandwidths of memory, but also of application bandwidth asymmetry and NVM bandwidth asymmetry.

In DRAM-cache-based systems:

- The bandwidth efficiency improves with an increase in the DRAM hit rate (and thus a decrease in NVM bandwidth utilization).
- As the DRAM hit rate increases, the ratio of read to write bandwidth demand on the DRAM cache increases, while the ratio of read to write bandwidth demand on NVM decreases.
- When the DRAM hit rate is low, all DRAM cache organizations perform equally in terms of bandwidth efficiency as they are all constrained by NVM bandwidth, obscuring differences in cache organization.
- When NVM bandwidth or the DRAM hit rate is very low, the most cost-effective memory system should be used as it has no negative impact on bandwidth efficiency.
- When NVM bandwidth is high enough, more bandwidth-efficient DRAM cache organizations should be considered as they deliver higher bandwidths at high hit rates.
- NVM write bandwidth has a significant impact on delivered read bandwidth since it affects DRAM utilization. This motivates the need for conducting research on techniques to minimize write bandwidth demand on NVM.

Our models can be extended in many aspects. Hit-miss predictors in the processor can affect both access latency and DRAM bandwidth efficiency. We plan to extend our models to include predictors. In addition, we plan to include separate DRAM hit rates for reads and writes in our models in order to more accurately characterize application behaviors.

Acknowledgments

We would like to thank Alex Breslow and Gabriel Loh for their comments. 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.

References

[1] "High bandwidth memory (HBM) DRAM JESD235," 2013. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd235

[2] "Hybrid memory cube specification 2.0," 2014. [Online]. Available: http://hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.0_Public.pdf

[3] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page placement strategies for GPUs within heterogeneous memory systems," in *ASPLOS*, 2015, pp. 607–618.

[4] E. Bolotin, D. Nellans, O. Villa, M. O'Connor, A. Ramirez, and S. Keckler, "Designing efficient heterogeneous memory architectures," *IEEE Micro*, vol. 35, no. 4, pp. 60–68, July 2015.

[5] D. Callahan, J. Cocke, and K. Kennedy, "Estimating interlock and improving balance for pipelined architectures," *J. of Parallel and Distributed Computing*, vol. 5, no. 4, pp. 334–358, 1988.

[6] C. Chou, A. Jaleel, and M. Qureshi, "CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *Micro*, 2014, pp. 1–12.

[7] ——, "BEAR: Techniques for mitigating bandwidth bloat in gigascale DRAM caches," in *ISCA*, 2015, pp. 198–210.

[8] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Symp. on VLSI Technology (VLSIT)*, June 2012, pp. 87–88.

[9] D. Jevdjic, G. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," in *MICRO*, 2014, pp. 25–37.

[10] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? Have it all with footprint cache," in *ISCA*, 2013, pp. 404–415.

[11] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-centric system interconnect design with hybrid memory cubes," in *PACT*, 2013, pp. 145–155.

[12] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. Lee, "A fully associative, tagless DRAM cache," in *ISCA*, 2015, pp. 211–222.
[13] G. Loh and M. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *MICRO*, 2011, pp. 454–464.
[14] ——, "Supporting very large DRAM caches with compound-access scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, May 2012.
[15] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hotchips*, 2011.
[16] M. Qureshi, M. Franceschini, A. Jagmohan, and L. Lastras, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times," in *ISCA*, 2012, pp. 380–391.
[17] M. Qureshi and G. Loh, "Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design," in *MICRO*, 2012, pp. 235–246.
[18] M. Radulovic, D. Zivanovic, D. Ruiz, B. R. de Supinski, S. A. McKee, P. Radojkovic, and E. Ayguadé, "Another trip to the wall: How much will stacked DRAM benefit HPC?" in *Proc. Intl. Symp. on Memory Systems (MEMSYS)*, 2015, pp. 31–36.
[19] D. Roberts, A. Farmahini-Farahani, K. Cheng, N. Hu, D. Mayhew, and M. Ignatowski, "NMI: A new memory interface to enable innovation," in *Hotchips*, 2015.
[20] A. Sodani, "Knights Landing: 2nd generation Intel "Xeon Phi" processor," in *Hotchips*, 2015.
[21] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.
[22] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, Feb 2015, pp. 476–488.
[23] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *HPCA*, Feb. 2013, pp. 282–293.