Opportunities for Processing Near Non-Volatile Memory in Heterogeneous Memory Systems

Zhenhong Liu, Amin Farmahini-Farahani*, Nuwan Jayasena*
University of Illinois at Urbana–Champaign, *AMD Research, Advanced Micro Devices, Inc.
zliu118@illinois.edu, {afarmahi,nuwan.jayasena}@amd.com

Abstract

Growth of data volumes in application domains such as high-performance computing, machine learning, and data analytics places increasing demands on main memory capacity of computing systems. Emerging byte-addressable, non-volatile memory (NVM) technologies can enable increased main memory capacities to meet these application demands more cost-effectively than is possible with DRAM alone due to their superior cell densities and lower cost-per-bit. However, NVM delivers lower memory bandwidth and incurs higher access latency than DRAM, especially for writes. As a result, system organizations that incorporate NVM typically use it as a large, "second-level" memory in addition to a smaller pool of DRAM. In this work, we have evaluated and compared the performance of different processing near memory systems using DRAM and NVM. Our simulation results show that, for several classes of applications with large datasets that cannot fit in DRAM, there can be significant amounts of data movement between DRAM and NVM. Therefore, depending on application characteristics, processing near NVM can improve energy efficiency and performance over processing near DRAM despite the limitations of NVM such as lower bandwidth and higher write energy. We show that processing near NVM can be up to a few orders of magnitude more energy-efficient (based on energy-delay product) than processing near DRAM when the DRAM capacity can only accommodate a fraction of the application's dataset.

1. Introduction

With rapid growth in data volumes, the capacity of the main memory should be scaled up to accommodate data-intensive applications. Emerging non-volatile memory (NVM) technologies promise higher density and lower cost-per-bit than volatile DRAM counterparts without incurring leakage energy, providing a promising avenue for cost-effectively meeting the increased memory capacity requirements of applications. However, NVMs typically suffer from lower bandwidth and higher access latency (especially for writes) as well as limited write endurance. To get the best of both worlds, heterogeneous memory systems combine DRAM and byte-addressable NVM to balance design requirements of the memory system such as bandwidth, capacity, and cost.

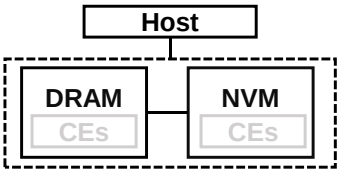

Figure 1. Abstract diagram of a heterogeneous memory system composed of DRAM and NVM.

At the same time, the end of Dennard scaling and the slowdown in Moore's law are necessitating architectural innovations to process big data workloads in an energy-efficient and high-performance manner. One such innovation is processing near memory (PNM), which can improve performance and energy efficiency of data-intensive applications by reducing communication between host processor and memory, and leveraging high local memory bandwidth [1, 2]. Maturing die-stacking technology and tools enable 2.5D and 3D integration of logic and memory dies in close proximity to realize PNM.

Broadly stated, PNM can help two classes of applications. The first are applications with large datasets and/or very low data locality that do not benefit from the host's caches. For these applications, moving data through the off-chip memory interface and the host's deep cache hierarchy degrades energy efficiency without commensurate performance improvements. PNM, instead, eliminates the energy overhead of moving data by processing data near where it is stored. The second class of applications are those with low compute intensity (low FLOPs/byte) that do not fully utilize host's compute capabilities. Running these applications on the host leads to underutilization of host's compute resources while memory bandwidth is highly stressed. PNM can take advantage of higher internal memory bandwidth and more energy-efficient internal compute resources[1].

Although PNM sets the stage for high-performance, energy-efficient data processing for a wide and important group of

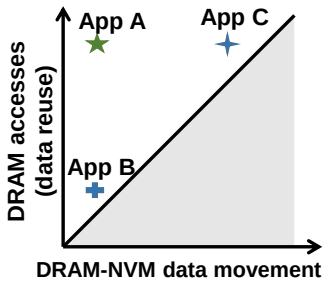

Figure 2. Application examples for processing near memory.

[1] Near-memory compute engines are typically less aggressive in their implementations than mainstream processors, improving their energy efficiency for memory-bound applications.

data-intensive applications, the benefits and challenges of PNM in heterogeneous memory systems composed of DRAM and NVM have not been investigated. To enable PNM in such a system (Figure 1), near-memory compute engines (CEs) can be envisioned near DRAM, near NVM, or even both. The primary open question is: what applications can achieve further performance or energy efficiency improvement by processing near NVM over processing near DRAM? As an example, application type A in Figure 2 is possibly more amenable to processing near DRAM as near-DRAM CEs access and reuse data multiple times once data is initially loaded in DRAM. As a result, the overhead of data movement between DRAM and NVM is amortized over several accesses to high-bandwidth DRAM. On the other hand, application types B and C in Figure 2 are possibly more amenable to processing near NVM. The reason is, if processed near DRAM, the amount of data movement between DRAM and NVM would be relatively close to the amount of data reuse. This paper aims to answer this question by examining the tradeoffs between processing near DRAM and processing near NVM, and makes the following contributions:

- We identify and categorize several characteristics that make applications amenable to processing near DRAM and/or processing near NVM (Section 3).
- We evaluate and discuss the performance and energy efficiency of applications under different PNM schemes and configurations (Sections 4-6).
- We demonstrate that several factors related to application characteristics (e.g., data reuse and read to write ratio) as well as heterogeneous memory configuration (e.g., DRAM-NVM bandwidth ratio and DRAM capacity) affect whether or not processing near NVM outperforms processing near DRAM.

We finally highlight the fact that processing near NVM, even though potentially challenging, can improve the performance and energy efficiency of an important group of applications by obviating the need for multiple data movements between DRAM and the higher-capacity NVM.

2. Background

Processing near NVM presents new benefits and challenges compared to processing near DRAM. The differences between these two types of PNM schemes arise from their important differences in the way data is partitioned, moved in and out of memory devices, and processed near memory. We elaborate on their differences in this section.

For applications with large datasets, data partitioning is highly dependent on available memory capacity. NVMs typically exhibit better memory cell density and technology node scalability than DRAM devices and thus can provide higher memory capacity in the same area. This denotes that a larger portion of the application's dataset can fit in a single NVM device compared with a DRAM device. To process applications with large datasets near DRAM, their datasets should be partitioned into smaller chunks that can fit in DRAM. Processing the same application near NVM will require fewer, larger partitions or, in some cases, no partitioning at all (if the entire application dataset fits in one NVM device). Thus, data partitioning is either not required or simpler to implement for processing near NVM. This is especially important for irregular applications (e.g., graph processing), for which achieving a data partitioning that considers load balance and communication overhead is quite challenging [3, 4].

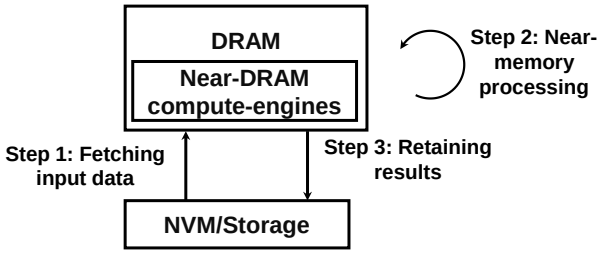

Figure 3. High-level steps of processing near DRAM.

Processing application data near DRAM requires different steps from processing near NVM since the memory hierarchies are different. Figure 3 shows three high-level steps that should be performed to process data near DRAM. First, data is brought in to DRAM from NVM (step 1). Next, data is processed near DRAM (step 2). Finally, results are written back from DRAM to NVM (step 3). Steps 1 and 3 involve data movement, while step 2 involves actual data processing. These three steps may partially be overlapped depending on the programming model used and application behavior or may be performed in sequence due to inter-dependency, hurting application performance. The performance of steps 1 and 3 could be constrained by the NVM bandwidth and DRAM-NVM interface bandwidth. Moreover, steps 1 and 3 incur energy consumption due to intra-chip data movement within NVM and DRAM devices, and inter-chip data movement over the off-chip DRAM-NVM interface. The data movement energy overhead is especially significant for streaming applications and others that touch each piece of data only once prior to replacing it with a new one in DRAM. On the other hand, processing near NVM eliminates steps 1 and 3 entirely as data is processed in-situ.

The actual data processing (step 2) can be performed faster and more energy-efficiently near DRAM due to its higher internal bandwidth and lower access energy. This benefit of near-DRAM processing is more notable for applications in which data is reused multiple times in short distances from the first access. Short reuse distance in applications provides the opportunity to initially load DRAM with a chunk of data, process it locally without frequently moving data between DRAM and NVM, and then switch to another chunk of data. This way the data movement overhead (steps 1 and 3) is amortized over large amounts of processing near DRAM. For applications with short reuse distances, since NVM has usually lower memory bandwidth than DRAM, the percentage of memory bandwidth utilization increases in processing near NVM. Hence, some applications that are compute-bound in

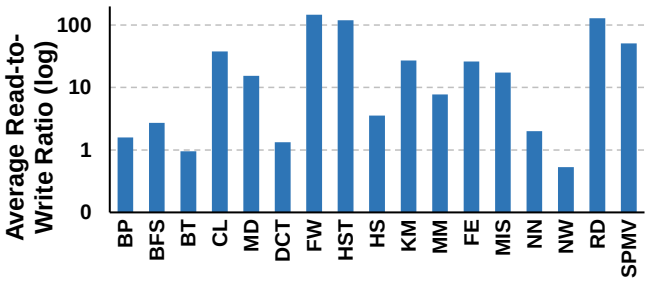

Figure 4. The average read-to-write ratio.

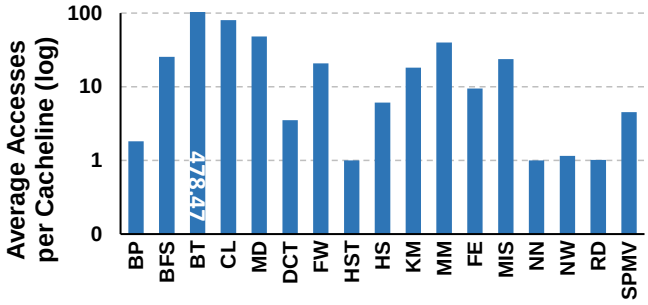

Figure 5. Average number of accesses per cacheline.

near-DRAM processing may become memory-bandwidth-bound when processed near NVM, provided the same compute engines are used.

However, if an application exhibits high data reuse but with long reuse distances, processing near NVM would potentially deliver better performance as data could be displaced from DRAM prior to reuse.

DRAM and NVM have different memory attributes. NVMs typically have lower write bandwidth, higher write latency, and present endurance challenges, while DRAM has symmetrical read and write latency/bandwidth and virtually infinite write endurance. Although the write bandwidth itself is typically not critical to overall performance, writes can occupy memory and decrease the available read bandwidth. Thus, writes that cannot be filtered by the cache hierarchy may cause performance degradations and endurance challenges. There have been several hardware optimizations to decrease the number of writes to NVM cells [5, 6, 7, 8, 9, 10] and software techniques to lower write bandwidth requirements of applications [11]. However, writes should be still considered in designs for processing near NVM.

3. Application Characterization

In this section, we study a wide range of applications with different characteristics. Our goal is to evaluate the amenability of these applications to near-DRAM processing and near-NVM processing based on their memory characteristics. The evaluated applications are listed in Table 1. To do this study, we generate and use memory traces of application kernels. More details about the traces can be found in Section 5.

To study application characteristics, we define and measure three metrics: read-to-write ratio, data reuse, and data reuse distance. These metrics can be used to categorize whether an

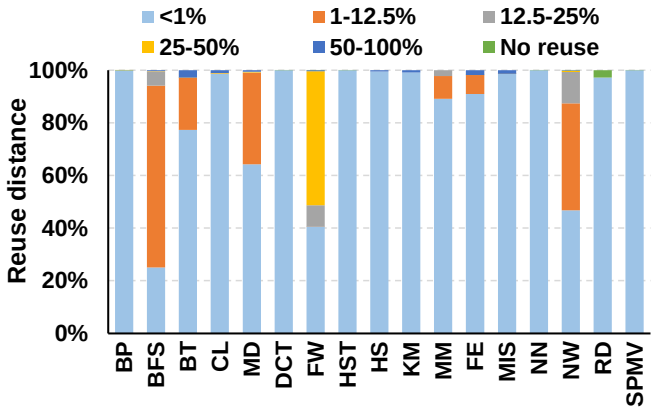

Figure 6. Reuse distance relative to the data set size.

application is more suitable for processing near DRAM or NVM. Note that for some applications, we have reduced the number of kernel iterations to limit the size of memory traces so we can use them for simulation in a reasonable time. Therefore, some measurements could slightly change with a higher iteration count.

Read-to-Write Ratio: Since NVMs usually have higher read bandwidth than write bandwidth, intuitively, applications with higher read-to-write (R2W) ratios are more suitable for near-NVM processing. Therefore, one should consider the R2W ratio of applications when evaluating them for near-NVM processing. We define the average R2W ratio as $\frac{\#\ of\ memory\ read\ requests}{\#\ of\ memory\ write\ requests}$. Figure 4 shows the average R2W ratio across our evaluated applications. Some of the applications show very high R2W ratios, up to about 150. For applications with high R2W ratios, the read bandwidth is the dominant factor in performance. Therefore, such applications are potentially good candidates for near-NVM processing.

Data Reuse: We use data reuse (along with the reuse distance) to characterize how efficiently the application dataset can be cached in DRAM. We use the average number of data accesses per cacheline to measure data reuse as shown in Figure 5. Some applications such as BP, HST, NW, and RD have a small number of accesses or even no data reuse (one access per cacheline), while some applications such as BT and CL have a very large number of accesses per cacheline. Applications showing significant data reuse are more suitable for near-DRAM processing when the capacity of DRAM is large enough to capture their working sets. However, when the working set size exceeds DRAM capacity, even data with a lot of reuse may need to be fetched multiple times from NVM depending on DRAM capacity and data reuse distance, which may impact the performance significantly.

Data Reuse Distance: We use data reuse distance to show how well the application dataset fits into DRAM when the DRAM capacity is limited. Reuse distance is a relatively accurate metric to predict the effectiveness of data caching [12]. Since data movement between DRAM and NVM is performed at page granularity (4KB), we measure the reuse distance at page granularity. Data reuse distance is defined as the

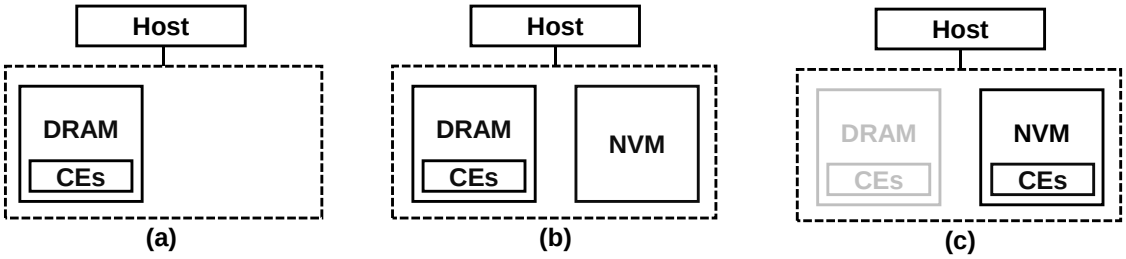

Figure 7. Evaluated PNM schemes: Idealistic processing near DRAM (a), processing near DRAM (b), processing near NVM (c).

number of distinct pages referenced in between two consecutive references to the same page. The reuse distance of a page denotes how much DRAM capacity is needed to hold the page in DRAM until the next reference to it. Therefore, applications with high data reuse and large reuse distances may suffer significantly from processing near DRAM when the DRAM capacity is limited as the same data elements have to be fetched repeatedly. On the other hand, applications with high data reuse, but small reuse distances can benefit significantly from processing near DRAM.

Figure 6 shows the reuse distances of consecutive references to the same page in terms of the number of distinct pages referenced in between (back-to-back references to the same page are excluded). The reuse distance is represented as the relative size of the application dataset. For example, for an application whose dataset includes 1000 pages, page references with a reuse distance of 25-50% have 250-499 distinct pages referenced in between those references. Applications with small reuse distance such as CL can be effectively cached in DRAM with a small capacity. Applications with large reuse distance such as FW and NW can severely suffer from a small DRAM capacity. Note that even when there is a single request whose distance is larger than the DRAM capacity, the DRAM needs to evict and fetch a whole page. Therefore, a relatively small number of such requests can severely impact performance.

4. PNM Schemes

The system that we evaluated consists of a host processor, a DRAM stack and an NVM stack. A memory stack is a 3D organization of memory dies and a single logic die. The near-memory CEs are implemented in the logic die. We assume CEs are GPUs as their user-programmability and high thread-level parallelism (and thus high bandwidth utilization) make them a good match for processing a wide variety of highly parallel and memory-intensive applications that are amenable to PNM [13]. The insights provided to us from this work is also valid for other types of throughput-oriented CEs such as field-programmable gate arrays (FPGAs) and custom accelerators as our analysis is not based on the detailed architecture of the underlying CEs.

Figure 7 shows abstract diagrams of three different schemes for PNM that we investigate in following sections. Note that we do not include comparisons against processing by the host as the memory-intensive applications that would typically be executed using PNM do not benefit from the host's cache hierarchy and greater compute power. Also, we defer schemes in which both DRAM and NVM have concurrent PNM capabilities to future work as the performance of such schemes depend highly on data partitioning and synchronization between the two and poses programming challenges. Further, we focus this work on schemes with a single DRAM stack and a single NVM stack. Inter-node communication in systems with multiple stacks each of DRAM and NVM is an important topic but is also deferred to future research.

4.1 Idealistic Processing Near DRAM

In this scheme, DRAM constitutes the main memory address space and uses in-stack CEs for PNM, as shown in Figure 7a. This organization resembles the conceptual near-DRAM processing systems, but we (unrealistically) assume that the capacity of a single DRAM stack is large enough to hold the entire application's dataset and the entire dataset is loaded into the DRAM prior to application execution. Thus, this shows an idealistic scheme for processing near DRAM which provides an upper bound on performance.

4.2 Processing Near DRAM

In this scheme, shown in Figure 7b, DRAM as the high-bandwidth memory is used for PNM, while NVM as the high-capacity memory is used only for storing application's dataset. This scheme resembles near-DRAM processing systems where DRAM is used as working memory backed up by NVM. In this scheme, only CEs integrated with the DRAM stack are used for data processing. Input data needed by near-DRAM CEs is first brought into DRAM from NVM before it can be processed. But, dynamic memory buffers are allocated in DRAM directly, and data are brought in from NVM only when it requires the data from NVM to initialize the buffer.

Data movement between DRAM and NVM is orchestrated at 4KB page granularity as finer grain data management incurs high metadata storage and management overhead [14]. To eschew the high write latency, write energy, and endurance challenges in NVM, we do not swap pages between DRAM and NVM. Instead, DRAM is software-managed and we copy pages from NVM to DRAM upon request. Only dirty data (not necessarily the whole page) is written back to NVM upon page eviction. This policy not only improves the performance of applications that cause frequent evictions of clean pages from DRAM, but also reduces writes to NVM. We model the overhead of data movement between the two.

Table 1. List of benchmarks studied.

Benchmark	Suite	Acronym
Back propagation	Rodinia	BP
Breadth-first search	Rodinia	BFS
Bitonic sort	AMD SDK	BT
Graph coloring	Pannotia	CL
Codesign molecular dynamics	Mantevo	MD
Discrete cosine transform	AMD SDK	DCT
Floyd-Warshall shortest path	Pannotia	FW
Histogram	AMD SDK	HST
Hotspot	Rodinia	HS
K-means clustering	Rodinia	KM
Matrix Multiplication	AMD SDK	MM
Mini finite element	Mantevo	FE
Maximal independent set	Pannotia	MIS
Nearest Neighbor	Rodinia	NN
Needleman-Wunsch	Rodinia	NW
Reduction	AMD SDK	RD
Sparse Matrix-Vector Multiplication	OpenDwarf	SPMV

4.3 Processing Near NVM

This scheme, as shown in Figure 7c, uses near-NVM CEs to enable processing near NVM. In this scheme, we assume that the capacity of NVM is large enough to hold the entire application's dataset. The CEs in the DRAM stack are not used in this scheme.

5. Experimental Methodology

Application Workloads: We used benchmarks from Rodinia [15], Pannotia [16], OpenDwarf [17], AMD Application SDK [18] and Mantevo [19] suites. Table 1 lists all the benchmarks we evaluated.

System Configuration and Memory Traces: In our system shown in Figure 7, the host CPU processor and GPU (CEs) share physical memory. We have generated memory traces using an in-house version of the gem5-gpu simulator [20] to investigate applications' memory characteristics. The control-intensive portion of the applications are run on the CPU and data-parallel kernels are run on the GPU. In the traces, we only capture the memory requests of the kernels running on the GPU as we assume only the kernels would be processed near memory. The near-memory GPU has 16 compute units (CUs), with each CU having a 64-wide SIMD pipeline, to resemble CEs integrated in the logic layer of a 3D-stacked memory considering thermal and area constraints shown by prior work [21, 13]. The near-memory GPU and its cache hierarchy are modeled after AMD's Graphics Core Next architecture [22]. Each CU has a 16KB private L1. We assume that near-memory CEs do not have L2 caches as they are unlikely to benefit memory-intensive applications suitable for PNM. Thus, memory traces are captured after L1 caches and requests serviced by the L1 caches are not included in the traces.

There are a few applications in Table 1 that are traditionally considered as memory-bandwidth-intensive but require relatively low bandwidth in our configuration. These applications are CL, HST, HS, MM, and RD. CL has two major kernels, one memory-intensive and one non-memory-intensive. The average bandwidth appears relatively low. HS and MM use scratchpad memory in the CUs to capture input data reuse which lowers their required main memory bandwidth. HST and RD also use scratchpad memory to store and reduce intermediate results. They also use memory barriers which further lower the frequency of memory accesses. Therefore, these applications are not memory-bandwidth-intensive in this configuration.

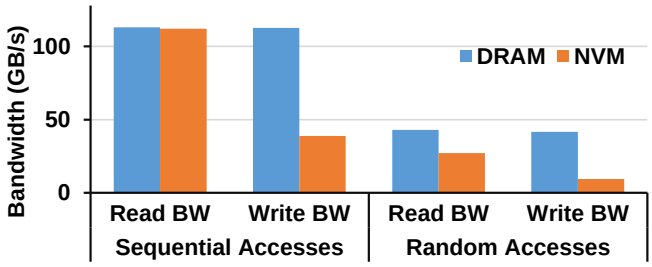

Figure 8. DRAM and NVM bandwidths.

Simulation Infrastructure: To evaluate the performance of our PNM schemes, we use the trace-driven memory simulator Ramulator [23]. We have added our NVM model to it and have enhanced it by using timestamps in the memory traces.

DRAM and NVM Models: In our models, both DRAM and NVM architecture and interface are modeled after High Bandwidth Memory (HBM) [24]. Both DRAM and NVM use a 1024-bit data bus interface at 1Gbps, have 8 channels and 8 banks. DRAM uses a 2KB row buffer. We use phase-change memory (PCM) technology as a representative of NVM. We assume that NVM is byte-addressable and has a 256-byte row buffer [25, 26, 27]. We have adopted the NVM timing parameters from Xu et al. [25].

We run synthetic tests to measure the sequential and random bandwidths of read and write accesses for our DRAM and NVM models. Sequential accesses are used to determine the bandwidth under maximum row buffer hit rate, while random accesses are used to determine the bandwidth under virtually no row buffer hits. In our experiments, memory request addresses are sliced in the order of Row, Bank, Column, and Channel, from MSB to LSB to increase channel-level parallelism and row buffer hit rate. Since data is cacheline-interleaved among channels, sequential accesses utilize all channels, reaching near theoretical peak bandwidth. Figure 8 compares DRAM and NVM bandwidths for sequential accesses and random accesses. The sequential read and write bandwidth of DRAM is 113.08GB/s and 112.18GB/s, respectively. Our NVM model has almost the same sequential read bandwidth as DRAM as most requests hit in row buffers and NVM has the same column access latency (tCL) as DRAM. However, NVM has much lower write bandwidth and random access bandwidth due to higher row activation delay (tRCD) and write recovery time (tWR).

The footprints of our applications can be very large in real deployments. To refrain from long simulation times, we use smaller datasets (e.g., 100MB) and configure DRAM capacity

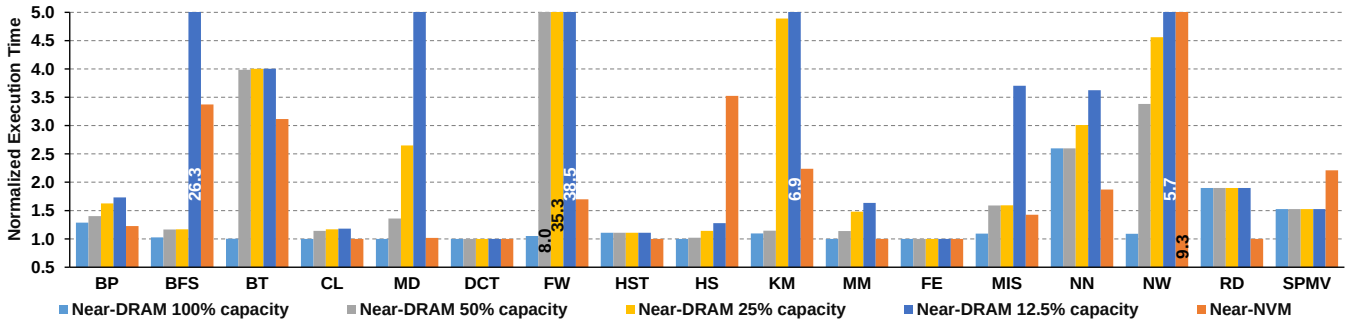

Figure 9. Execution time normalized to the *idealistic processing near DRAM* scheme.

according to the memory footprint of each application in our simulation. We assume the capacity of NVM is large enough to hold the entire dataset. The purpose of configuring DRAM capacity is to model the cases when DRAM does not provide enough capacity while NVM does, which is the scope of this work. Considering the likely large capacity gap between DRAM and NVM in realistic systems with both memory types, such an assumption is reasonable.

Energy: We estimate energy consumption of memory stacks and interfaces using a high-level energy model. We assume that DRAM core access and wire traversal within the DRAM die are 4pJ/b [28] and 0.8pJ/b [29], respectively. DRAM background and refresh power is estimated at 10% of the maximum active power of the DRAM dies [30]. We assume NVM read energy is 11.82pJ/b and NVM write energy is 369.02 pJ/b [31]. We assume the data transfer over TSV consumes 0.4pJ/b on average for an 8-high die-stack [32]. We use serial interfaces for off-chip communication between DRAM and NVM. We assume that off-chip interface and its logic consume 4.5pJ/b [33, 34]. We also assume interface power consumption is constant regardless of utilization as idle symbols need to be transmitted [35, 36].

6. Results

6.1 Performance

We evaluate the performance of the three schemes described in Section 4 to find out what applications benefit from near-NVM processing. Figure 9 compares the execution time of the three schemes normalized to that of the *idealistic processing near DRAM* scheme. For the *processing near DRAM* scheme, we use DRAM with different capacities to show the impact of DRAM capacity on the performance: we evaluate DRAM capacity of 100%, 50%, 25% and 12.5% of each application's dataset. When the DRAM capacity is 100% of the application's dataset, the applications incur only cold misses for initial fetch of input data from NVM. All subsequent accesses to the data are retrieved from DRAM. As we mentioned in Section 4, NVM contains the entire application's dataset.

Figure 9 shows that processing near-NVM provides promising performance for a large number of applications. The *processing near NVM* scheme can even achieve the same performance as the *idealistic processing near DRAM* scheme for some applications (CL, MD, DCT, HST, MM, FE, and RD). We can explain this using three characteristics of the applications: R2W ratio, bandwidth requirement, and access pattern. First, these six applications, except DCT, have a R2W ratio greater than 5, which mitigates the negative impact of the NVM's low write bandwidth. Second, these six applications, except CL and RD, require a relatively low memory bandwidth, which means that the bandwidth provided by NVM can exceed their bandwidth requirements. Finally, these applications have regular access patterns which leads to relatively high row buffer hit rate, especially CL and RD. Having a high hit rate allows applications to take advantage of NVM's limited bandwidth more efficiently. This is especially important for CL and RD as their required bandwidth approaches the NVM's maximum.

Comparing the *processing near NVM* scheme with the *processing near DRAM* scheme, the former still can outperform the latter for many applications. Even when the DRAM capacity is large enough for the whole application's dataset (DRAM capacity is 100% of application dataset), processing near NVM for BP and NN outperforms the *processing near DRAM* scheme by 5% and 39%, respectively. The reason is that, although these two applications have relatively low R2W (less than or equal to 2.0), they have little or no data reuse. Therefore, the performance overhead of fetching data from NVM is not amortized over multiple accesses in DRAM, degrading the performance of processing near DRAM.

However, deploying DRAM with the same capacity as NVM is highly unlikely due to cost. When compared with limited-capacity DRAM, more applications benefit from the processing near NVM scheme. When the DRAM capacity becomes as small as 12.5% of the application dataset, most applications perform better using the *processing near NVM* scheme. This highlights the fact that data movement between DRAM and NVM can significantly degrade performance of the *processing near DRAM* scheme when DRAM capacity is limited. However, for HS, NW, and SPMV, near-DRAM processing still outperforms near-NVM processing even using DRAM with limited capacity. HS involves a considerable number of writes at the end of its execution which cannot be effectively overlapped with other operations, making processing near NVM slower than processing near DRAM. NW has extremely low R2W ratio (0.53). Hence, it severely suffers from the NVM's low write bandwidth. SPMV is a highly

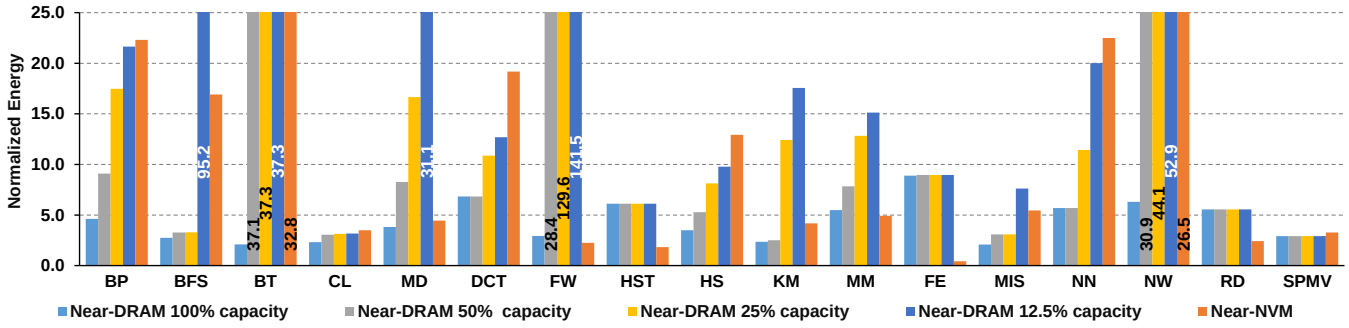

Figure 10. Energy consumption normalized to the *idealistic processing near DRAM* scheme.

memory-intensive application with relatively high data reuse in short distances, making it a good fit for near-DRAM processing. The data reuse in SPMV is mostly caused by L1 cache thrashing. We expect that an optimized implementation of SPMV with reduced cache thrashing could improve its performance for near-NVM processing.

Insights: We can roughly group the applications into two categories according to their performance degradation with the decrease in the DRAM capacity: (1) applications such as BP and SPMV whose performance does not degrade or degrades slightly, and (2) applications such as BFS and FW whose performance degrades sharply. Such different behaviors are mainly caused by applications' differences in data reuse distance. When the data reuse distance is smaller than the DRAM capacity, the referenced data can be effectively cached in DRAM. However, when the reuse distance is larger than the DRAM capacity, the referenced data is evicted from DRAM before its subsequent use and must be fetched from NVM again. In addition, when DRAM capacity is limited, for applications such as BP, BFS and NN a large amount of *dirty* data is evicted to NVM, which degrades the performance significantly.

6.2 Energy and Energy Efficiency

We evaluate the energy consumption of memory stacks, memory interface, and DRAM-NVM interconnect for different PNM schemes. Figure 10 shows the energy consumption of the schemes normalized to that of the *idealistic processing near DRAM* scheme. Since processing near NVM does not entail consuming refresh energy and DRAM-NVM interconnect static energy, it has a significant advantage in terms of energy for compute-intensive applications with high R2W ratio such as FE, MM, HST, and RD. For other applications, the *processing near NVM* scheme generally consumes more energy than the *processing near DRAM* scheme when the DRAM capacity is large enough to capture most of application's dataset as NVM has higher access energy per bit, especially writes. However, when the DRAM capacity becomes smaller, the energy advantage of near-DRAM processing diminishes quickly as more data movement happens between DRAM and NVM. When the DRAM capacity is 12.5% of application's dataset, a considerable number of applications consume lower energy for processing near-NVM. The exceptions are BP, CL, DCT, HS, NN, and SPMV in which the *processing near NVM* scheme consumes slightly more energy than the *processing near DRAM* scheme with 12.5% capacity. This is mainly because these six applications have relatively low R2W ratio.

We further use the energy-delay product as a metric for energy efficiency. Compared with *processing near DRAM* with 12.5% capacity, *processing near NVM* provides better energy efficiency for all applications, except DCT, HS, and SPMV. The energy efficiency of FE on *processing near NVM* is even better than that on *idealistic processing near DRAM*. The reason is that our implementation of FE is greatly compute-intensive. Thus, background energy of DRAM is the dominant portion of memory energy. Although NVM has higher access energy than DRAM, the background energy spent on DRAM makes the total energy of DRAM higher.

Insights: The energy and energy efficiency results show that when the dataset of the application is larger than the DRAM capacity, additional DRAM and NVM accesses and data movement between the two offset the access energy and bandwidth advantage of DRAM over NVM. In such cases, processing near NVM can avoid those additional memory accesses and improve the energy efficiency of the system.

7. Conclusions

Many workloads with large datasets such as in-memory databases, data analytics, and scientific computing can be processed more efficiently near NVM than DRAM given the entire dataset can fit into one level of memory, enabling significant energy saving by obviating the need to move data between different levels of memory hierarchy. In this work, we compare the performance of processing near lower-capacity DRAM with that of processing near higher-capacity NVM for a wide range of applications. In terms of performance, the following classes of computations are amenable to processing near NVM:

- Computations that access data only once or few times such as those that perform pre-processing or post-processing (e.g., histogram and reduction).
- Computations with data reuse but a reuse distance larger than the DRAM capacity such as those that touch large datasets iteratively.
- Computations with large datasets that are difficult to partition into smaller chunks due to their irregular access patterns (e.g., graph processing).

- Computations with high read-to-write ratios or whose writes can be effectively overlapped with other operations.

In terms of energy, computations with high read-to-write ratios that cannot be effectively cached in DRAM are amenable to processing near NVM due to eliminating DRAM refresh energy and DRAM-NVM interface energy.

Acknowledgments

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] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro (Special Issue on Big Data)*, vol. 34, pp. 36–43, 2014.

[2] A. Farmahini-Farahani, K. Morrow, J. Ahn, and N. Kim., "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *HPCA*, 2015, pp. 283–295.

[3] L. Wang, Y. Xiao, B. Shao, and H. Wang, "How to partition a billion-node graph," Tech. Rep. MSR-TR-2013-102, Feb. 2013. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=183714

[4] D. Margo and M. Seltzer, "A scalable distributed graph partitioner," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1478–1489, Aug. 2015.

[5] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *HPCA*, Feb. 2013, pp. 282–293.

[6] 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.

[7] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *HPCA*, 2010, pp. 1–11.

[8] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "FPB: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Micro*, 2012, pp. 1–12.

[9] B. D. Yang, J. E. Lee, J. S. Kim, J. Cho, S. Y. Lee, and B. G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *IEEE Intl. Symp. on Circuits and Systems*, 2007, pp. 3014–3017.

[10] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *MICRO*, 2009, pp. 347–357.

[11] S. Chen, P. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *CIDR*, 2011, pp. 21–31.

[12] K. Beyls and E. H. D'Hollander, "Reuse distance as a metric for cache behavior," in *Conf. on Parallel and Distributed Computing and Systems*, 2001, pp. 617–662.

[13] D. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented programmable processing in memory," in *Proc. of Intl. Symp. on High-performance Parallel and Distributed Computing*, 2014.

[14] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Intl. Conf. on Supercomputing*, 2011, pp. 85–95.

[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009, pp. 44–54.

[16] S. Che, B. M. Beckmann, S. K. Reinhard, and K. Skadron, "Pannotia: Understanding irregular GPGPU graph applications," in *IISWC*, 2013, pp. 185–195.

[17] W.-C. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 dwarfs: A work in progress," in *Proc. of Intl. Conf. on Performance Engineering*, 2012, pp. 291–294.

[18] "AMD accelerated parallel processing (APP) software development kit (SDK)." [Online]. Available: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

[19] M. Heroux, D. Doerfler, P. Crozier, J. Willenbring, H. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, and R. Numrich, "Improving performance via mini-applications," SAND2009-5574, Tech. Rep., 2009.

[20] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, 2015.

[21] Y. Eckert, N. Jayasena, and G. Loh, "Thermal feasibility of die-stacked processing in memory," in *Workshop on Near-Data Processing*, 2014.

[22] AMD, "White paper: AMD graphics cores next (GCN) architecture," Jun 2012.

[23] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2015.

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

[25] 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.

[26] J. Meza, J. Li, and O. Mutlu, "A case for small row buffers in non-volatile main memories," in *ICCD*, 2012, pp. 484–485.

[27] ——, "Evaluating row buffer locality in future non-volatile main memories," Carnegie Mellon University, Tech. Rep., 2012.

[28] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Micro*, 2010, pp. 363–374.

[29] *International Technology Roadmap for Semiconductors, 2011 Edition*, 2012 update.

[30] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *ISPASS*, 2014, pp. 190–200.

[31] 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 Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 40:1–40:25, Dec. 2014.

[32] "ITRS interconnect working group, 2012 update," www.itrs.net/links/-2012Summer/Interconnect.pptx.

[33] 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.

[34] G. Sandhu, "DRAM scaling and bandwidth challenges," in *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, 2012.

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

[36] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple*, 1st ed., 2005.