Online Management of Hybrid DRAM-NVMM Memory for HPC

Reza Salkhordeh and André Brinkmann
Johannes Gutenberg University
Mainz, Germany
Email: {rsalkhor, brinkman}@uni-mainz.de

***Abstract*—Non-volatile main memories (NVMMs) offer a comparable performance to DRAM, while requiring lower static power consumption and enabling higher densities. NVMM therefore can provide opportunities for improving both energy efficiency and costs of main memory. Previous hybrid main memory management approaches for HPC either do not consider the unique characteristics of NVMMs, depend on high profiling costs, or need source code modifications.**

In this paper, we investigate HPC applications' behaviors in the presence of NVMM as part of the main memory. By performing a comprehensive study of HPC applications and based on several key observations, we propose an online hybrid memory architecture for HPC. It only requires low-overhead sampling of memory accesses for its page placement decisions. The experimental results obtained through running a wide range of HPC applications show that the proposed architecture can service up to 88% of accesses from DRAM if 90% of the main memory is built from NVMM. The NVMM lifetime can also be extended by up to 90% compared to all-NVMM.

***Keywords*-HPC; non-volatile memory; heterogeneous memory; performance;**

I. Introduction

The main memory architecture has become one of the most challenging aspects when designing high-performance computers. DRAM as standard memory technology is facing the scalability wall and it is getting more and more difficult to produce denser chips [23]. DRAM also significantly contributes to the total energy consumption of HPC environments [8]. Therefore, main memory is turning into a bottleneck which hinders the further scalability of HPC centers.

The ever-increasing demand of HPC applications for more computational power and higher memory capacities as well as these scalability issues force manufacturers and the HPC community to pursue alternative architectures beyond DRAM.

Non-volatile main memory (NVMM) is an umbrella term covering a number of upcoming memory technologies which do not need constant cell refreshing to sustain their voltage levels. Example technologies are phase-change memory (PCM), racetrack memory, resistive random-access memory (ReRAM), or Intel's 3D XPoint technology [25], [29]. They enable significant reductions of the static energy consumption and also can be manufactured in denser chips [16]. NVMMs are therefore expected to have lower costs than DRAM [1]. Most NVMM technologies can only endure a limited number of writes and have an asymmetric read/write performance. They therefore cannot simply replace DRAM as main memory [23]. Without *wear-leveling* techniques, NVMMs can have lifetimes as low as one month [9].

To take advantage of NVMMs promising characteristics and to mitigate their shortcomings, several approaches explored architectures that employ both DRAM and NVMMs as main memory. Hybrid main memories typically try to place *hot* data pages in the faster memory, while *cold* data pages are moved towards slower memory [1], [15].

DRAM-NVMM architectures are shown to be effective in reducing the power consumption and costs with minimal performance impact [1], [30]. Such architectures, however, are mostly not designed for HPC environments and cannot, due to their complex algorithms, be employed without significant hardware modifications. The few previously proposed DRAM-NVMM architectures for HPC operate by first profiling complete application runs and then statically decide the data placement for future runs [5], [26]. This approach is not portable and can only be used in few predefined scenarios. To the best of our knowledge, none of the previous studies have simultaneously considered NVMM characteristics, online profiling without the need for a prior analysis, and the applicability to HPC programs without source modifications.

In this paper, first we present insights into the effect of running HPC applications on NVMM devices. Second, we propose a management scheme for hybrid memories in HPC, which decides the data page placement based on the workload and NVMM characteristics. In brief, the main contributions of the proposed scheme are as follows:

1) For the first time, we present a comprehensive study on memory access patterns of HPC workloads, considering various applications, sampling rates, and CPU cache levels.
2) We propose guidelines for lowering the memory access profiling overhead without any need for specialized hardware, OS, or libraries. To the best of our knowledge, none of the previous architectures has such a broad applicability.
3) Unlike previous studies, our strategy does not need costly a priori application runs to profile data accesses.
4) The proposed hybrid main memory architecture identifies performance critical data pages by sampling and moves

them to DRAM. Both costly writes to NVMM and its limited lifetime are considered for the data placement.

5) Data pages are grouped into *bins* to significantly reduce the memory overhead with negligible accuracy loss.

Three modules are employed: a profiler, a workload characterization unit, and a migration manager. The profiler collects data on the access pattern of data pages. The workload characterization computes the impact of data pages on overall system performance and identifies data pages that need to be moved between memories. Finally, the migration manager moves the selected data pages between DRAM and NVMM.

The experimental evaluation was conducted on an NVMM emulator, embedded into a real HPC environment with several nodes. We also compared our architecture with state-of-the-art approaches requiring hardware modifications using simulations [30]. The comparison shows that the new architecture is able to achieve a very similar data page placement quality, while needing orders of magnitude less resources.

II. Related Work

The significant impact of main memory on overall system performance and power consumption has motivated many researchers to investigate *hybrid* memory architectures to improve main memory efficiency.

Several studies aimed to reduce the latency gap between the CPU's LLC and main memory by employing *faster than DRAM* memories as DRAM cache [2], [39]. In many use-cases, the interface between the CPU and the memory module is a performance bottleneck and stacked DRAM can be placed within the CPU chip to remove it. The limited on-chip size of stacked memory restricts its usage to caching [2].

Banshee reduces in-chip DRAM cache replacement traffic by modifying the TLB, the memory controller, and by introducing a bandwidth-aware frequency-based cache replacement policy [39]. A framework to identify and move hot memory allocations to stacked DRAM is suggested in [31], which requires an initial full profiling run of the application for the identification process. This shortcoming also occurs in other hybrid memory architectures (see, e.g., [3]). These studies are orthogonal to our work, since we want to expand the main memory size by using slower NVMM, while maintaining good performance. On-chip DRAM can be used on top of the hybrid DRAM-NVMM main memory to further hide latencies.

Several studies examined the integration of NVMMs into main memory as an extension of DRAM [18], [19], [24], [30], [35]. They can be categorized into general hybrid DRAM-NVMM architectures, explorations of NVMMs for HPC applications, and the simulation of hybrid memories.

The general approaches provide placement approaches for data pages independent of the application type and the overall system architecture. Park et. al. tried to reduce the DRAM refresh energy in hybrid architectures by modifying the DRAM controller [24]. Such architectures require controller modifications and are not within the scope of this paper. CLOCK-DWF places read-dominant data pages in NVMM and moves them to DRAM when they receive write requests [18]. Its migration costs are significant, since any write access to NVMM results in moving the data page to DRAM. Implementing the modifications within the CLOCK algorithm also requires hardware modifications. TwoLRU analyzes the performance, power, and lifetime costs of migrating data pages between memories and limits the number of migrations [30]. TwoLRU imposes significant hardware overheads for its LRU implementation. M-CLOCK tries to reduce the high migration costs between memories, while also reducing the number of NVMM writes [17]. Similar to CLOCK-DWF, two clock handles are employed in M-CLOCK and a *lazy* bit is added to the CLOCK algorithm. A write access to a page in NVMM, which has its *lazy* bit set, results in migrating it to DRAM. Updates to the CLOCK data structure are conducted by the CPU and M-CLOCK therefore requires CPU modifications. UIMigrate also tries to reduce the number of migrations by considering the migration cost [33]. It employs a global data structure for identifying hot pages in both DRAM and NVMM. To control migrations, pages are moved from NVMM to DRAM if their score is more than that of a page in DRAM plus the migration cost.

HeteroOS shows that virtualization platforms can also benefit from hybrid memories if the hypervisor has access to the guest OS data structures and can employ them to steer the data page placement [12].

The second group of previous studies focused on optimizing hybrid memories for HPC applications. LWPTool profiles applications to identify loops and memory-intensive data structures [38]. They are fed into an offline analyzer to compute the placement of data objects. CoMerge profiles applications to identify performance-critical data objects and places them on DRAM [5]. In addition to the LWPTool, it optimizes the interplay of multiple applications running on the same node by considering the applications' sensitivity to moving data to NVMM. The need for the offline phase and multiple runs of the application significantly reduces the applicability of LWPTool and CoMerge. The HpMC memory controller sits between the LLC and the memory bus. It switches between an inclusive cache-based architecture, where the DRAM serves as cache for the NVMM, and an exclusive flat memory systems based on the temporal locality of the workload. HpMC requires a redesign of the memory controller to either use NVMM as memory extension or as cache [32].

There are several studies on hybrid memories in HPC which require source code modifications to steer the data page placement [7], [20], [26], [34]. Static decision using

profiling data for placing data objects in either DRAM or NVMM have been shown to be effective for several applications [26]. Unimem enables the programmer (and requires from him) to annotate compute-intensive loops and target data objects, while allocating memory with a custom allocator [34]. Unimem profiles the application, based on the annotations and then calculates the data placement. Migrating data between memories is conducted via a separate thread to reduce the performance overhead. UH-MEM estimates the benefit resulting from migrating data pages from NVMM to DRAM for HPC applications and pages with higher overall system benefit are chosen for migration [19]. The performance benefit of moving a data page is computed by considering parameters such as access frequency, data locality, and possible parallelism of servicing the requests. Hardware as well as software modifications are required to implement such an architecture. These studies, however, are not in the scope of this paper, as their applicability is limited to use-cases where the source code of the program is available.

Tahoe [35] has been designed for task-parallel programs and profiles a set of representative tasks and computes suitable data pages for placing in DRAM. The memory page placement is determined before running the tasks, based on the output of the representative tasks. In case of limited DRAM while running multiple tasks, data pages are selected for migration to DRAM based on the predicted performance effect. Unlike Tahoe, the architecture proposed in this paper is designed for general HPC environments, where no such profiling is possible.

Due to the lack of access to NVMM devices, previous studies proposed emulators to analyze NVMMs. HME reserves a NUMA node for emulating NVMM and analyzes the number of memory accesses to this NUMA node for each processor core [6]. After fixed intervals, it adds delays to the applications by keeping physical processes busy. The delays are based on the number of remote DRAM accesses, which acts as NVMM replacement, and the difference between remote DRAM and target NVMM latencies. The same approach has been proposed in pVM [13]. Siena has been designed to allow fast design space explorations to analyze many possible memory configurations and usages. It transforms the application behavior and the machine model into a domain specific language (DSL) which can be simulated using a memory simulator [27].

Additional aspects of hybrid memories such as reliability and durability have been investigated in previous studies like [11], [36], which are also not in the scope of this paper.

To summarize, previous studies exploring DRAM-NVMM memories for HPC either suffer from applicability due to source code modification, require altering the hardware, or depend on full-run profiling.

III. HPC Workloads Characterization

This section conducts a comprehensive study on the memory behavior of HPC applications. The goal of this characterization is three-fold: study the feasibility of using hybrid memories for HPC applications by understanding HPC applications' access patterns, present the trade-off between memory access sampling overhead and its accuracy, and explore the effect of the profiling interval on the efficiency of hybrid memory.

We have selected five applications from the NERSC trinity benchmark set [4] and also included NAMD [28], Tealeaf [22], and Lulesh [14]. To ensure that this characterization is generalizable to a wide range of HPC environments, the workloads are selected in such a way to cover most types of HPC applications such as: 1) CPU or memory intensive (CI/MI) programs, 2) applications with large or small memory footprint (LM/SM), and 3) programs with high or low locality (HL/LL). We measured the number of memory accesses per executed instruction to determine CPU- or memory-intensive workloads. The memory footprint is defined as the number of unique accessed virtual memory pages. A program is categorized as having a high locality if more then 50% of accesses belong to the top 10% most accessed data pages. The applications can be characterized as follows:

- AMG (MI, LM, LL) is a multigrid solver for unstructured grids.
- MiniGhost (MI, LM, LL) simulates heat diffusions across homogeneous domains using a stencil approach.
- GTC (MI, SM, LL) is a parallel particle-in-cell code to support the plasma experiment reactor ITER.
- MILC (CI, SM, HL) is a quantum chromo dynamics application.
- NAMD (MI, LM, HL) is a molecular dynamics package scaling beyond 500.000 cores.
- SNAP (CI, LM, LL) mimics the behavior of the neutral particle transport simulator PARTISN.
- Tealeaf (CI, SM, HL) is a benchmark designed to explore the architectural design space for iterative sparse linear solvers.
- Lulesh (MI, SM, HL) approximates hydrodynamics equations by solving a simple Sedov blast problem. It can be seen as representative for C++-based HPC applications.

The applications were running on eight nodes using 32 cores each. Each node included two Skylake Xeon Gold 6130 16-core processors, offering 96 GByte memory. For this analysis, both instrumenting memory accesses by Intel PIN [21] and hardware sampling using *Processor Event-Based Sampling* (PEBS) [10] have been used.

Intel PIN instrumented all memory operations of the programs including libraries such as MPI. Most memory operations are cached within the different CPU caches and never reach the main memory. It is therefore necessary to

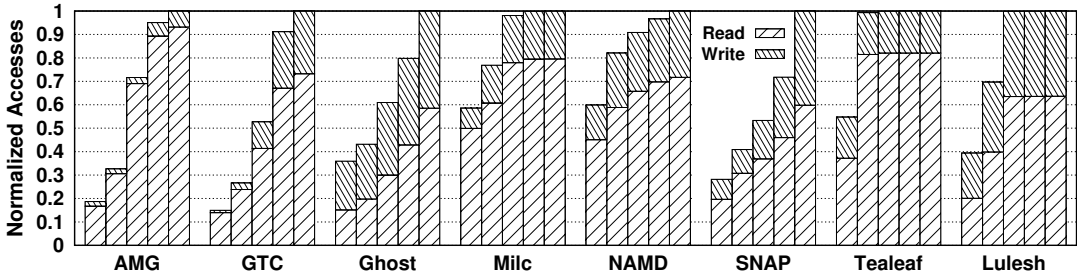


Figure 1: Normalized number of accesses, which can be serviced by DRAM ratios of 5%, 10%, 25%, 50%, and 100%.

simulate the cache levels to obtain accesses from PIN traces that have been actually sent to memory. We have tried to keep the simulation as similar as possible to the physical Skylake CPU.

In PEBS, the processor stores its state in a memory buffer at a given sample rate. The OS sets a counter in the processor alongside the required operation (i.e., memory load) to sample. The processor decreases the counter by one each time the operation is executed. When the counter reaches zero, the processor captures its internal state in a memory buffer specified by the OS. Once the buffer is full, the processor informs the OS via an interrupt to process the data. Using this method significantly reduces the sampling performance overhead, since the OS is only involved each time the buffer is full.

A. Applicability of DRAM-NVMM for HPC

DRAM is (still) significantly faster and can endure more write accesses than higher-density NVMM. Hybrid DRAM-NVMM main memories are therefore most effective when a large percentage of accesses can be serviced by a small set of data pages. Moving these *hot* data pages to DRAM enables us to achieve a performance-efficient hybrid memory, while reducing overall power consumption and NVMM wear-out.

We analyzed the HPC applications and investigated the access patters to their working sets. Fig. 1 shows the normalized maximum number of accesses, which can be serviced by various DRAM ratios within hybrid memory, if all accesses are known in advance. We can make four major observations:

- A DRAM capacity of 25% of the working set size is always sufficient to service more than half of the accesses from DRAM. This shows that hybrid memory can be effectively deployed in HPC.
- Most accesses to *hot* pages are reads and reserving DRAM for write-intensive pages can under-utilize DRAM.
- The selected HPC applications are read-intensive, which is the preferred workload type for hybrid memories.
- Data writes would mostly be serviced by NVMM if only data hotness would be considered for *GTC* or *SNAP*.

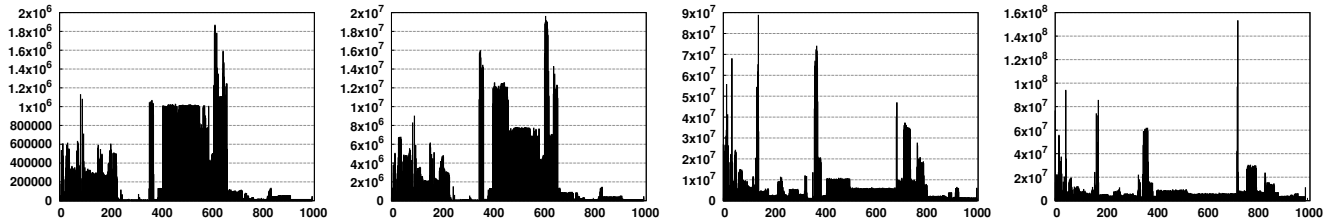
B. Sampling Overhead and Accuracy Tradeoff

The hybrid memory management scheme proposed in this paper requires an online profiler to identify performance-critical data pages. The online profiler, of course, incurs performance overheads when processing a huge number of samples. Obtaining all memory accesses in the runtime is therefore impractical due to the significant overheads and it is necessary to minimize the overhead of the profiler. Reducing the sample rate can significantly decrease the performance overhead and allows us to perform a more complex analysis on the sampled memory accesses, while aiming for less than 1% performance overhead. The accuracy of sampling, however, depends on having a sufficient number of sampled memory accesses in a fixed time interval.

We used the Intel PIN tool to extract the access frequencies to memory pages over a range of sampling rates, as PEBS imposes a limit on the maximum sampling rate. The preprocessing of the (complete) data allowed us to virtually scale the sampling rates from including all accesses up to only having a granularity of one sample per one million accesses.

Fig. 2 shows the access frequencies to memory pages for *AMG* including various sampling rates. Please note that based on different sampling rates, the scaling differs between the experiments. To normalize the figures, the number of sampled pages are multiplied by the sampling rate. All *accessed* virtual addresses were sorted and grouped into 1,000 equally-sized bins. To do so, we pass through the sorted list of pages and mark the start and the end address of each bin. Note that only data pages, which have been accessed are considered. The effect of the number of bins is discussed later in this section.

Lower sample rates cannot sample all virtual addresses and therefore the boundaries of the bins as well as the peaks might be shifted to the left or the right, compared to sampling all accesses. Another observation is that additional peaks are present for lower sample rates. Our analysis has shown that this is due to numerous consecutive accesses to *hot* data pages, which results in sampling such data pages regardless of the sample rate. If such anomalies are ignored, hot regions can be identified even for one sample per 64k



(a) All accesses (b) Once per 16 accesses (c) Once per 16k accesses (d) Once per 64k accesses

Figure 2: Memory access frequency of *AMG* for various sampling rates over 1000 bins

Table I: Parameters Description

Parameter	Description
a_k^α	Sampling accuracy for sample rate k and DRAM size of α
h_k^α	Set of α hottest data pages for sample rate k
w_h	The weight of old priority of bins. Defaults to 0.1
lat_r	NVMM relative write latency, compared to DRAM
lat_w	NVMM relative read latency, compared to DRAM
t_x	Execution time of phase x
CPU_x	CPU usage of phase x

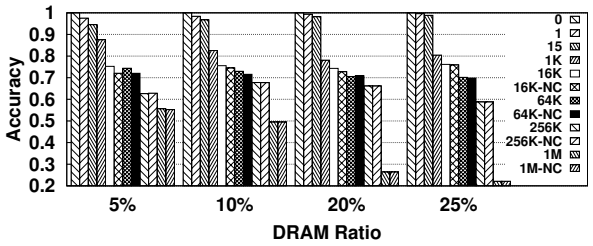


Figure 3: Sampling rates accuracy for AMG benchmark

accesses. The hot regions are, however, flattened for lower sample rates.

The sampling accuracy a is defined as denoted in Equation 1. It depends on the number α of the hottest pages (which will be moved to DRAM) and the sampling rate k. The description of notations throughout the paper is reported in Table I. Sampling all accesses leads to the most precise identification of hot data pages, and hence, it is used as the baseline to measure the accuracy of other sampling rates. To this end, the hottest data pages identified by k sample rate (h_k^α) is compared to the identified pages by the sampling of all accesses (h_1^α). The percentage of correctly selected data pages by k sampling rate (pages present in both h_k^α and h_1^α) is considered as the accuracy for this sampling rate.

$$a_k^\alpha = \frac{|h_1^\alpha \bigcap h_k^\alpha|}{|h_1^\alpha|} \quad (1)$$

Fig. 3 shows the accuracy of various sample rates for choosing hot data pages. The x-axis denotes the percentage of the selected data pages and the y-axis denotes the accuracy according to Equation 1. *NC* sample rates denote *No Cache* values, where the simulation of the CPU cache levels has been turned off and all load and store operations are considered to directly access main memory. Obtaining all memory operations for sampling rates higher than once per 16k accesses is not considered due to the significant performance overhead of simulating the accesses without cache.

Sampling at least once per 64k requests guarantees for our application mix a minimum accuracy of 70%, which seems acceptable, considering its negligible performance impact. Note that although a lot of information is lost for this sampling rate, it does not need to correctly predict the *order* or *access frequency* of hot data pages and as long as a hot data page is moved to DRAM by a sampling rate, it is considered as a correct hot page identification. Decreasing the sample rate to less than once per 64k results in losing the boundaries of hot regions and hence, our architecture cannot use such sample rates. Due to the page limitation, we omitted the results for the other benchmarks, while their results lead to the same conclusions.

The effect of simulating CPU cache levels on the sampling accuracy is also shown in Fig. 3. Considering all sampled load and store operations as memory accesses has a negligible effect on the overall accuracy. Therefore, for both instrumenting and hardware-based simulation, sampled load and stores can be used instead of the actual memory accesses. The hardware-based sampling also benefits from this simplification, since sampling all types of memory accesses (e.g., prefetch, dirty page eviction, or read miss) is not possible for several CPUs. Additionally, sampling memory operations can be done without root privilege, which increases its applicability.

Fig. 4 shows the accuracy for a varying number of bins. Grouping the pages into 1.000 bins results in more than 80% accuracy compared to having one individual counter for every data page. Increasing the number of bins to 10.000 slightly improves the accuracy, while it adds 10x more memory usage. Using less than 1.000 bins cannot provide sufficient accuracy and the memory usage saving cannot justify the accuracy loss.

The actual performance overhead of sampling memory accesses is evaluated by running applications using various

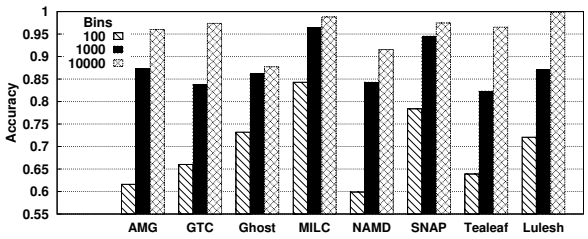


Figure 4: Accuracy of various bins count

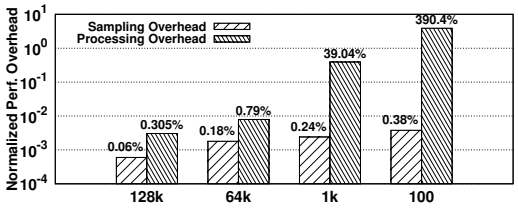


Figure 5: Performance overhead for various sampling rates

sampling rates via PEBS. Fig. 5 shows the additional CPU usage for these sampling rates. Sampling once per 64k accesses has less than 0.18% performance overhead per processor core for the sampling and additional 0.79% overhead for processing the samples. Note that although sampling at high frequencies such as once per 100 accesses has less than 0.5% performance overhead, the amount of samples that needs to be processed by the proposed architecture significantly increases: one sample per 100 accesses leads to an overhead of 390%. Therefore, sampling once per 64k accesses seems suitable from both accuracy and overhead.

C. Sampling Interval

Sampling memory accesses can impose performance overheads to the system. In this section, we evaluate the effect of only sampling a fixed number of accesses and deciding the placement of data pages based on such short-time sampling. Pages are, in this scenario, only allowed to be moved once between DRAM and NVMM. To this end, the number of DRAM accesses is compared in two scenarios:

1) The optimal scenario profiles the whole runtime of an application by running it once for profiling, deciding the placement and running it again counting DRAM accesses,
2) The online scenario profiles a fixed number of accesses at the beginning of the execution, performs data movements based on this profiling and counts DRAM accesses.

The first scenario has the shortcomings of previous studies, which require running the application at least twice and is only used as a baseline. Furthermore, all data pages in Scenario 2 are placed in NVMM first. Fig. 6 shows the normalized average memory access time (AMAT) for varying numbers of profiled accesses. The x-axis denotes the DRAM-percentage of the overall memory. In this experiment, data pages are only moved *once* between memories. All values are normalized to the *optimal* data placement, which knows all accesses to the data pages *in advance*. It migrates pages with highest number of accesses in the future to DRAM. We used a fixed number of accesses for profiling, since we assume that we do not know the type of the running application. Hence, it is not possible to set the profiling interval as a percentage of the approximate total number of accesses of the application. If such information is available, they can be easily integrated into our architecture to further improve accuracy or profiling time.

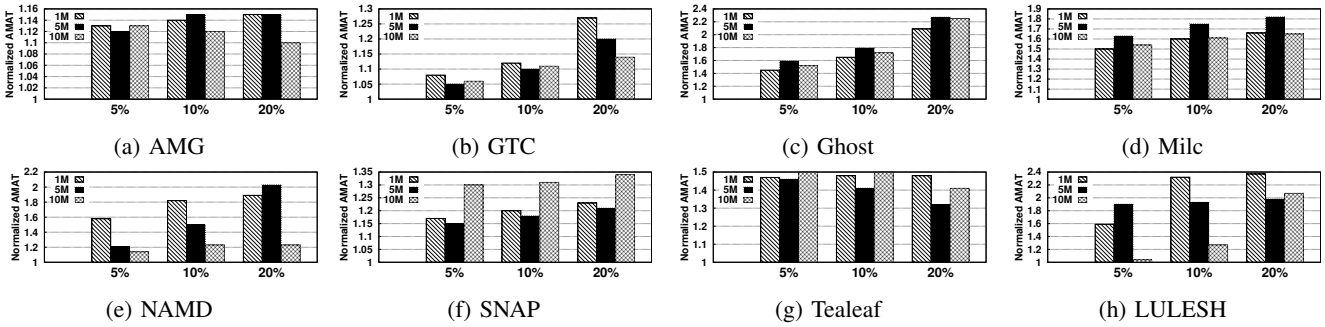
One observation is that increasing the profiling interval does not necessarily result in performance improvements. This is because data pages are moved to DRAM later if the profiling interval is longer, compared to short profiling intervals. Another observation is that very short profiling intervals like 1 mil. accesses can provide comparable performance to the *optimal* algorithm if the DRAM size is relatively small. This is due to the distribution of accesses, where a few data pages are massively accessed. This distribution can be easily identified by all algorithms and profiling intervals if the DRAM capacity is small. Profiling 5 mil. accesses seems to be a suitable profiling interval for most of the examined HPC applications, in terms of profiling overhead and provided latency.

The main conclusions of the comprehensive workload characterization are as follows:

1) Hot memory regions can be identified with high accuracy by sampling once per 64k accesses.
2) Simulating CPU cache levels does not have a significant positive effect on the overall accuracy and can be omitted.
3) Grouping memory data pages into 1.000 bins can significantly reduce the memory overhead, while imposing negligible accuracy loss.
4) Profiling a number of accesses at the start of the HPC applications can be employed to identify *hot* data pages with high accuracy. A fixed value, suitable for most of the HPC applications, can be selected.
5) A relative short profiling phase can compete with an optimal placement if the DRAM size is relatively small compared to the NVMM size.

IV. Proposed Architecture

The proposed hybrid DRAM-NVMM main memory architecture considers the observations made in the previous section to accurately perform data page placements. It is designed in a way that *no* hardware or OS modifications are required. All employed libraries are packages shipped with standard Linux distributions. This ensures that the proposed hybrid memory management scheme has a higher portability compared to previous approaches. This is important, since NVMMs has just entered the market, and the manufacturers did not adapt any of the previous ideas, which



(a) AMG (b) GTC (c) Ghost (d) Milc

(e) NAMD (f) SNAP (g) Tealeaf (h) LULESH

Figure 6: Normalized AMAT of various number of profiled accesses (1M/5M/10M) compared to an optimal hybrid memory.

require hardware modifications. Compatibility with common operating systems and standard libraries employed in HPC environments ensures a fast and cost-efficient integration of NVMMs. To reduce the memory overhead and to ensure that the proposed architecture is scalable in terms of memory size, the address space is split into a fixed number of bins and housekeeping information is only kept per bin.

Fig. 7 shows the overall information flow of the proposed architecture. Dashed boxes are modules of the proposed architecture, while solid boxes are unmodified. Applications access virtual addresses, which are mapped to physical addresses on either DRAM or NVMM. The CPU samples the accessed memory locations and sends them to the *Profiler* module. PEBS records are decoded in this module and the accessed virtual addresses are passed to the *Characterization* module. After deciding which data pages need to be migrated, their addresses are sent to the *Migration* module. This module prepares the required data and calls a system call to actually move the data pages. In the following, the three new modules of the proposed architecture are presented in more detail.

Section III-C has shown that the access pattern can be well predicted by only sampling the memory accesses at the beginning of a *short-running* application and then by moving data pages according to their hotness. Longer running applications can of course change their access pattern and the proposed approach therefore works in phases of configurable length. Profiling, characterization, and migration are again only performed at the beginning of each phase to minimize their performance impact.

A. Profiler

The profiling module samples the memory accesses of the target application. Accesses from the OS and other software such as monitoring and job scheduling are discarded by the profiler. This results in a lower number of sampled accesses and hence, reduces the performance overhead.

The workload characterization in Section III has shown that we can sample memory operations instead of the actual memory accesses with only a negligible loss in precision.

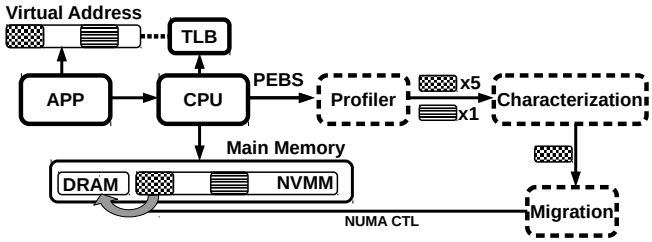


Figure 7: Architecture of the proposed hybrid memory

The profiler is able to use both techniques and if hardware sampling of memory accesses is not possible, we switch and sample memory operations. Instead of physical addresses, the proposed profiler collects virtual addresses of the HPC application. They are more portable and can be used in standard libraries for moving data pages. Sampled virtual addresses are placed in bins, which only store the accumulated number of accesses to the data pages belonging to them. The profiler also stores the start and end address of each bin to be able to map each virtual address to one bin. Reads and writes are separately counted for data pages.

When the required number of samples has been collected by the profiler, it is disabled to remove the performance overhead and also to free the CPU hardware performance counters. After the profiling phase, the counters can be employed to count the number of memory accesses to DRAM and NVMM to observe the efficiency of the proposed architecture. Upon starting of the next profiling phase, the counters will be again used for profiling. Re-configuring the PEBS counters to change from a profiling to an evaluation phase, and vice versa, is not a costly operation and requires negligible time to perform.

B. Characterization

The *characterization* module analyzes the sampled accesses to perform the data page placements. Due to the difference between the read and write latencies of NVMMs, we count number of read and write accesses separately. Write-intensive requests should be given higher priority, since we gain more benefit in terms of performance, NVMM

lifetime, and also energy consumption from placing them in DRAM. We therefore consider each write access as equivalent to *three* read requests, which is close to their latency difference, according to previous studies [37].

The hotness of data pages can change during the application runtime. We therefore perform several sampling phases to adapt to such changes and decrease the priority of all data pages before a new sampling phase begins. Therefore, the most recent sample phase always has a higher impact on the final priority of data pages, compared to older samples.

The proposed architecture needs the minimum and maximum virtual addresses to decide the virtual addresses belonging to bins. To do so, we observe the sampled addresses in each sampling period and if any address exceeds the maximum (or minimum) address of previous samples, the new addresses are placed in the first or last bin. The first sampling period is the exception, which requires all accessed pages to decide the initial start and end address of bins.

The granularity of data pages is significantly higher than the memory access granularity. Therefore, a migrated data page from NVMM to DRAM should have a larger number of accesses to compensate for the migration cost. This cost is also considered in the decision-making process before moving data pages. The latency of reading a page from NVMM and writing it to DRAM (and moving a page from DRAM to NVMM, if DRAM is full) is considered as the migration cost. In our evaluations, this latency is equivalent to executing consecutive load and store operations to read from NVMM and write to DRAM (and in opposite direction if needed). If the priority of a data page in NVMM is higher than the priority of a data page in DRAM plus the migration cost, it is considered for migration. This method prevents costly migrations that do not provide enough performance and/or lifetime benefit. Furthermore, decisions are always made on the granularity of bins.

Equation 2 denotes the formula for calculating the priority of a bin. lat_r and lat_w denote the relative latencies of NVMM for read and write operations, respectively. The weight value w_h controls the effect of the previous priority on the new calculated priority. This will ensure that previously *hot* data pages are given more chances to stay in DRAM.

$$prio_{new} = w_h \cdot prio_{old} + \sum^{pages} (reads \cdot lat_r + writes \cdot lat_w) \cdot (1 - w_h) \quad (2)$$

C. Migration

The *migration* module is responsible for moving data pages at the granularity of bins between memories. Since DRAM and NVMM are assigned to separate (virtual or physical) NUMA nodes, we can move data pages by their virtual addresses using *numactl* and the OS handles all the required modifications in the page table of the process as well as of the TLB after moving the data page. This significantly improves the portability of the proposed architecture, while also reducing the performance overhead, since the OS kernel is optimized to perform these operations.

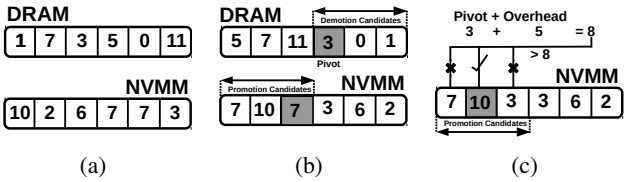


Figure 8: Selecting data pages for promotion/eviction. Numbers represent the priority of data pages.

The module first computes the bins that should be moved to the other memory, which is done in four steps:

1) Find the *K* bins in DRAM with the lowest priorities. Since we only need to partition the bins in DRAM, the complexity of the method is lower than that of sorting the bins ($n \cdot \log(n)$, where n denotes the number of bins in DRAM). The complexity of the partitioning is O(n) on average.
2) The *migration* module searches for bins in NVMM with higher priority, compared to the partition pivot of the DRAM bins. Note that the priority of the partition pivot is increased by the migration cost, before being compared to the NVMM bins. This ensures that the migration cost is considered for selecting bins to be promoted from NVMM to DRAM.
3) The required number of bins is moved from DRAM to NVMM to free-up space for the selected bins for promotion.
4) Bins from NVMM are promoted to DRAM, using a system call to the NUMA kernel module.

All steps are transparent to the application and there is no need to stop the execution of the application and/or block its access to a part of the data pages. Note that this method also works for shared data pages between application processes. If a shared data page is selected for promotion to DRAM, the virtual to physical address mapping of all other processes is also updated. Fig. 8a shows sample bins along with their priorities. The value of *K* is considered 3 in this example. Fig. 8b denotes the state of the partitioned bins after Step 1. Note that the bins within a partition are not necessarily sorted. The partitioning is conducted on both DRAM and NVMM, where the *K* lowest and highest priorities are selected from DRAM and NVMM, respectively. The priority of promotion candidates in NVMM is then compared to the DRAM partition pivot and eligible bins for promotion are selected as shown in Fig. 8c. The bins for demotion are selected from the partitioned bins. If we want to have a higher accuracy, we can sort such bins to select the bins with minimum priority. This, however, adds performance overhead to the system, if the number of partitioned bins

is high. Therefore, such sorting is optional in our proposed architecture.

The system call to move data pages can operate on non-continuous virtual addresses and hence, we can move all data pages of a bin by a single system call. The OS might not be able to move all data pages to NVMM due to reasons, such as locking. Since DRAM can hold many data pages, checking the status of each move can increase the performance overhead. Therefore, the proposed architecture does not retry moving data pages if the OS refused to move them. Such data pages remain in NVMM and if they are still *hot* in the next profiling cycle, the proposed architecture tries to move them to DRAM again. If the number of failed migrations is high, a part of the DRAM will be unused, which decreases the efficiency of the proposed architecture. In our evaluations, such locking rarely happened and therefore, we simply skipped the data pages that are not migrated to DRAM until the next phase.

V. Experimental Results

In this section, we first provide the experimental environment for our testing and then the evaluations of the proposed architecture is presented. We compared our architecture with a baseline DRAM-only, a baseline NVMM-only, TwoLRU and UIMigrate architectures [30], [33]. The experiments are conducted on the *MOGON II* supercomputer at the JGU Mainz. All nodes were equipped with two Intel Skylake Xeon Gold CPUs. The HME [6] emulator for NVMMs has been used to simulate NVMM. PCM is selected as the NVMM for evaluations since it is the most promising NVMM in terms of maturity. The read and write latencies of NVMM were set to 2x and 5x the latencies of DRAM, respectively. The energy consumption values are obtained from [30], since the values used in [33] are not known. Read and write energy consumption for DRAM and NVMM are $3.2\eta j$, $3.2\eta j$ and $6.4\eta j$, $32\eta j$, respectively. DRAM static power consumption is considered $\frac{1j}{GB \times second}$, while NVMM static power is one tenth of DRAM. The DRAM size was set to 10% of the working set size of the applications. The total memory size was varying between 1 GByte to 2.5 GByte per CPU core. Each phase lasted 10 minutes and the applications runtimes were varying between 1 to 6 hours.

A. DRAM Efficiency

Since DRAM has a better performance than NVMM, servicing more requests from DRAM also results in higher performance. To measure the efficiency of the proposed architecture, we measured the number of memory accesses serviced by DRAM. Fig. 9 shows the normalized number of DRAM accesses, compared to the total number of memory accesses. In most applications, more than 40% of the accesses are serviced by DRAM, which shows that the proposed architecture efficiently identified hot data pages for placing in DRAM.

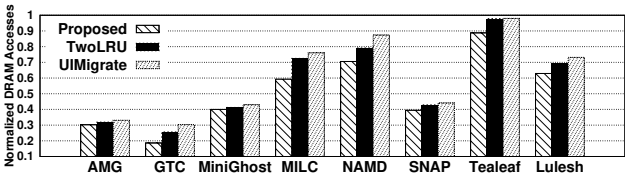


Figure 9: Normalized number of DRAM accesses, compared to all-DRAM main memory

One observation is that *Tealeaf* was able to service more than 90% of the requests by DRAM even if the accuracy of selecting bins has *only* been 80%. Our analysis has shown that this is due to the large number of bins with almost the same priority. The proposed architecture selected bins with negligible lower priority compared to the optimal algorithm. Such bins are identified as *incorrectly* selected, while the penalty to select them is small. Only 20% of the memory requests of the *GTC* benchmark were serviced by DRAM. The low DRAM utilization is based on the memory access pattern of this workload. Hot data pages do not have a spatial locality and by moving bins between memories, many cold data pages were also moved to DRAM. To prevent this problem, we can increase the number of bins for such workloads.

Fig. 9 also shows the normalized number of memory accesses, serviced by DRAM when using TwoLRU or UIMigrate. Both require hardware modifications and impose significant CPU overhead. The threshold values are set according to [30], [33]. The proposed architecture has a comparable performance to both, while removing the need for hardware and/or software modifications. The highest gap between the architectures can be seen for the *GTC* workload, since TwoLRU and UIMigrate manage the memory at the granularity of data pages, while our proposed architecture only requires a granularity on the level of bins. UIMigrate has higher DRAM efficiency, compared to TwoLRU, due to its superior migration control mechanism.

Fig. 10 shows the normalized execution time of the proposed architecture, compared to all-DRAM main memory. We do not compare our runtimes with TwoLRU and UIMigrate, as the use of them leads to extreme sampling and characterization overheads if not being implemented in hardware. Our proposed architecture significantly reduces the execution time of *all* benchmarks compared to all-NVMM systems by employing a small amount of DRAM in addition to the NVMM. For workloads like *NAMD*, the execution time is even reduced to half compared to all-NVMM. This shows the efficiency of hybrid memories and the effectiveness of the proposed architecture in identifying hot memory regions.

The comparison between all-DRAM and all-NVMM systems shows that the runtime difference can be as small as 30% for *GTC* or *AMG* workloads, while increasing to a

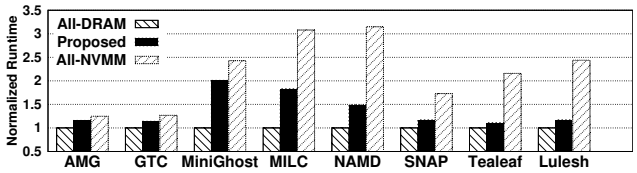


Figure 10: Normalized execution time, compared to all-DRAM main memory

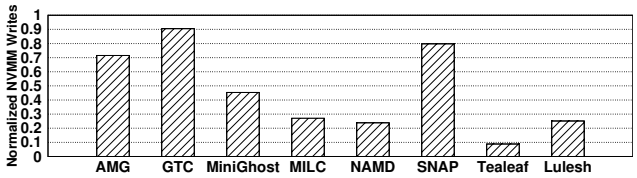


Figure 11: Normalized number of NVMM writes, compared to all-NVMM main memory

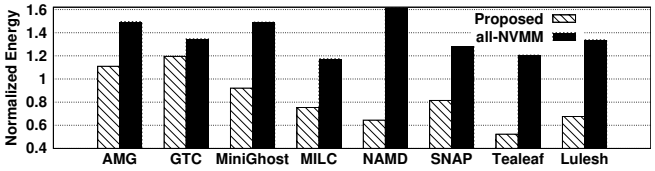


Figure 12: Normalized energy consumption, compared to all-DRAM main memory

factor of more than 3 for workloads like *MILC* or *NAMD*. This shows that employing slower memories does not affect all applications in the same manner and that using all-NVMM systems is a viable option for some workloads.

Our architecture only induces a performance penalty of less than 20% for most workloads when being compared to all-DRAM systems. Nevertheless, the benchmarks *MiniGhost*, *Tealeaf*, and *NAMD* show a higher sensitivity to the usage of NVMM. However, since NVMMs are more dense and less expensive than DRAM, we can have larger main memories and calculate more complex models per node *or* we can afford to build clusters with higher node counts.

B. NVMM Writes and Lifetime

Reducing the number of writes to NVMM can significantly benefit the HPC environment, due to the performance penalty of write accesses in NVMM and its limited lifetime. Fig. 11 shows the normalized number of write accesses to NVMM compared to an all-NVMM scenario. By moving 10% of the data pages to DRAM and by also prioritizing the write-intensive data pages, the proposed architecture reduces the number of NVMM writes by more than 50% in most of the examined workloads. This reduction in the number of writes results in extending the NVMM lifetime by 2x on average.

Tealeaf shows the highest reduction of writes to NVMM, as it has a relatively small set of write-intensive data pages. As a result, the NVMM lifetime can be extended by 10x compared to all-NVMM main memory. This shows the importance of prioritizing write-intensive data pages, instead of only considering the frequency of accesses. *GTC* shows the smallest reduction of writes to NVMM (only 10%), which is equal to the DRAM to NVMM ratio. This is, again, due to the low spatial locality of hot and write-intensive data pages in *GTC*, as we move data pages at the granularity of bins and therefore, in this case, also always move several cold data pages to DRAM.

C. Energy Consumption

Fig. 12 shows the normalized consumed energy of the proposed architecture compared to all-DRAM main memory. All-NVMM main memory fails to improve energy consumption due to its higher power consumption in read and write accesses. In *AMG* and *GTC* workloads, the hybrid architecture fails to save energy since most of the memory accesses are serviced by NVMM. Such workloads are not suitable for hybrid or all-NVMM architectures, in terms of energy saving. In other workloads, the proposed architecture has up to 48% energy saving, compared to all-DRAM main memory.

D. Architecture Overheads

The overheads of the proposed architecture can be categorized into three groups: 1) performance, 2) memory, and 3) CPU. The performance overhead concerns the increase in the runtime of the application. The performance overhead of profiling is reported in Fig. 5 for various sampling rates. The overhead for migrating data pages depends on the number of requested migrations between memories. In our experiments, the wall-clock time of executing the *move_pages* function was less than 2 seconds. However, not all accesses are blocked during the migrations and only the accesses to the moved data pages were serviced with slight delay. Note that the migration overhead is only dependent on the time intervals of profiling. If this interval is long enough (i.e., more than 2 minutes), which was the case in our experiments, the migration overhead will be less than 1%.

To decrease the memory overheads, we only obtain the number of accesses to bins, instead of all data pages. For each bin, four variables are stored: 1) start address, 2) end address, 3) number of reads, and 4) number of writes. Each variable had a size of 8 bytes and by considering 1.000 bins, the total overhead of managing bins was around 32KB. Hence, the total required memory for the proposed architecture is small enough to be placed in the L1 cache of the CPU, which minimizes its footprint on the overall system performance.

Equation 3 denotes the formula to prorate the CPU overhead of the various phases of the proposed architecture

across the execution of the application. t_x and CPU_x denote the time interval and CPU usage for phase x. The three phases are: 1) profiling ($prof$), 2) migrating data pages (mig), and 3) program execution after migrations (run). The CPU overhead of profiling is reported in Fig. 5. In the migration phase, where bin priorities are compared and candidates for migration are selected, the proposed architecture consumes 100% of the CPU time. The duration of this phase, however, is very short (less than 5 seconds) and hence, the overall CPU overhead is small. In the experiments, profiling is conducted every 10 minutes, and based on the experiments, the CPU overhead is less than 0.85% for all the examined workloads.

$$Overhead = \frac{t_{prof} \cdot CPU_{prof} + t_{mig} \cdot CPU_{mig}}{t_{prof} + t_{mig} + t_{run}} \quad (3)$$

VI. Conclusion

NVMMs can provide higher density and lower power consumption. Due to the asymmetric performance and endurance limitations, NVMMs cannot entirely replace DRAM. Hybrid DRAM-NVMM architectures have been suggested in previous studies, which suffer from several shortcomings such as high profiling overhead and need for source code modification. In this paper, we proposed an *online* hybrid memory management scheme, without any offline characterization and/or profiling. It groups data pages in bins to reduce the memory overhead and employs the sampling mechanism of CPUs to profile the workload in the runtime. Experimental results show that the proposed architecture can service 51% of accesses, on average, from a small DRAM. The number of writes serviced by NVMM is reduced by 46% on average, compared to all-NVMM.

References

[1] N. Agarwal and T. F. Wenisch, "Thermostat: Application-transparent page management for two-tiered main memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Xi'an, China, April 8-12*, 2017, pp. 631–644.

[2] L. Alvarez, M. Casas, J. Labarta, E. Ayguadé, M. Valero, and M. Moretó, "Runtime-guided management of stacked DRAM memories in task parallel programs," in *Proceedings of the 32nd International Conference on Supercomputing (ICS), Beijing, China, June 12-15*, 2018, pp. 218–228.

[3] H. Brunie, J. Jaeger, P. Carribault, and D. Barthou, "Profile-guided scope-based data allocation method," in *Proceedings of the International Symposium on Memory Systems (MEMSYS), Old Town Alexandria, VA, USA, October 01-04*, 2018, pp. 169–182.

[4] M. J. Cordery, B. Austin, H. J. Wassermann, C. S. Daley, N. J. Wright, S. D. Hammond, and D. Doerfler, "Analysis of cray XC30 performance using trinity-nersc-8 benchmarks and comparison with cray XE6 and IBM BG/Q," in *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation - 4th International Workshop (PMBS), Denver, CO, USA*, 2013, pp. 52–72.

[5] T. D. Doudali and A. Gavrilovska, "Comerge: toward efficient data placement in shared heterogeneous memory systems," in *Proceedings of the International Symposium on Memory Systems (MEMSYS), Alexandria, VA, USA, October 02 - 05*, 2017, pp. 251–261.

[6] Z. Duan, H. Liu, X. Liao, and H. Jin, "HME: A lightweight emulator for hybrid memory," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, March 19-23*, 2018, pp. 1375–1380.

[7] S. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems," in *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys), London, United Kingdom, April 18-21*, 2016, pp. 15:1–15:16.

[8] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, M. O'Connor, and O. Mutlu, "What your DRAM power models are not telling you: Lessons from a detailed experimental study," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (PMACS)*, vol. 2, no. 3, pp. 38:1–38:41, 2018.

[9] V. Gogte, W. Wang, S. Diestelhorst, A. Kolli, P. M. Chen, S. Narayanasamy, and T. F. Wenisch, "Software wear management for persistent memories," in *17th USENIX Conference on File and Storage Technologies (FAST), Boston, MA, February 25-28*, 2019, pp. 45–63.

[10] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, 2011.

[11] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. M. Tullsen, and R. K. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, Austria, February 24-28*, 2018, pp. 583–595.

[12] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "Heteroos: OS design for heterogeneous memory management in datacenter," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, June 24-28*, 2017, pp. 521–534.

[13] S. Kannan, A. Gavrilovska, and K. Schwan, "pvm: persistent virtual memory for efficient capacity scaling and object storage," in *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys), London, United Kingdom, April 18-21*, 2016, pp. 13:1–13:16.

[14] I. Karlin, J. McGraw, E. Gallardo, J. Keasler, E. A. León, and B. Still, "Abstract: Memory and parallelism exploration using the LULESH proxy application," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis (SC), Salt Lake City, UT, USA, November 10-16*, 2012, pp. 1427–1428.

[15] H. A. Khouzani, F. S. Hosseini, and C. Yang, "Segment and conflict aware page allocation and migration in DRAM-PCM hybrid main memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1458–1470, 2017.

[16] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, p. 143, 2010.

[17] M. Lee, D. Kang, and Y. I. Eom, "M-CLOCK: migration-optimized page replacement algorithm for hybrid memory architecture," *ACM Transactions on Storage (TOS)*, vol. 14, no. 3, pp. 25:1–25:17, 2018.

[18] S. Lee, H. Bahn, and S. H. Noh, "CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 2187–2200, 2014.

[19] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management," in *2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, September 5-8*, 2017, pp. 152–165.

[20] F. X. Lin and X. Liu, "*memif*: Towards programming heterogeneous memory asynchronously," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Atlanta, GA, USA, April 2-6*, 2016, pp. 369–383.

[21] C. Luk, R. S. Cohn, R. Muth, H. Patil, A. Klauser, P. G. Lowney, S. Wallace, V. J. Reddi, and K. M. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15*, 2005, pp. 190–200.

[22] S. McIntosh-Smith, M. Martineau, T. Deakin, G. Pawelczak, W. P. Gaudin, P. Garrett, W. Liu, R. P. Smedley-Stevenson, and D. Beckingsale, "Tealeaf: A mini-application to enable design-space explorations for iterative sparse linear solvers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, September 5-8*, 2017, pp. 842–849.

[23] O. Mutlu, "Memory scaling: A systems architecture perspective," in *Proceedings of Memcon, Santa Clara, CA, USA*, Aug. 2013.

[24] H. Park, S. Yoo, and S. Lee, "Power management of hybrid dram/pram-based main memory," in *Proceedings of the 48th Design Automation Conference (DAC), San Diego, California, USA, June 5-10*, 2011, pp. 59–64.

[25] S. S. P. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[26] A. J. Peña and P. Balaji, "Toward the efficient use of multiple explicitly managed memory subsystems," in *2014 IEEE International Conference on Cluster Computing (CLUSTER), Madrid, Spain, September 22-26*, 2014, pp. 123–131.

[27] I. B. Peng and J. S. Vetter, "Siena: exploring the design space of heterogeneous memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), Dallas, TX, USA, November 11-16*, 2018, pp. 33:1–33:14.

[28] J. C. Phillips, R. Braun, W. Wang, J. C. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. V. Kalé, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[29] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. Chen, R. M. S. and Martin Salinga, D. Krebs, S. C. and Hsiang-Lan Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4-5, pp. 465–480, 2008.

[30] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid DRAM-NVM memory architecture," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, March 14-18*, 2016, pp. 936–941.

[31] H. Servat, A. J. Peña, G. Llort, E. Mercadal, H. Hoppe, and J. Labarta, "Automating the application data placement in hybrid memory systems," in *2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, September 5-8*, 2017, pp. 126–136.

[32] C. Su, D. Roberts, E. A. León, K. W. Cameron, B. R. de Supinski, G. H. Loh, and D. S. Nikolopoulos, "Hpmc: An energy-aware management system of multi-level memory architectures," in *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS), Washington DC, DC, USA, October 5-8*, 2015, pp. 167–178.

[33] Y. Tan, B. Wang, Z. Yan, Q. Deng, X. Chen, and D. Liu, "Uimigrate: Adaptive data migration for hybrid non-volatile memory systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, March 25-29*, 2019, pp. 860–865.

[34] K. Wu, Y. Huang, and D. Li, "Unimem: runtime data managementon non-volatile memory-based heterogeneous main memory," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, (SC), Denver, CO, USA, November 12 - 17*, 2017, pp. 58:1–58:14.

[35] K. Wu, J. Ren, and D. Li, "Runtime data management on non-volatile memory-based heterogeneous memory for task-parallel programs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), Dallas, TX, USA, November 11-16*, 2018, pp. 31:1–31:13.

[36] J. Xu, L. Zhang, A. Memaripour, A. Gangadharaiah, A. Borase, T. B. D. Silva, S. Swanson, and A. Rudoff, "Nova-fortis: A fault-tolerant non-volatile main memory file system," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), Shanghai, China, October 28-31*, 2017, pp. 478–496.

[37] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Nimble page management for tiered memory systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Providence, RI, USA, April 13-17*, 2019, pp. 331–345.

[38] C. Yu, P. Roy, Y. Bai, H. Yang, and X. Liu, "Lwptool: A lightweight profiler to guide data layout optimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2489–2502, 2018.

[39] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: bandwidth-efficient DRAM caching via software/hardware cooperation," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Cambridge, MA, USA, October 14-18*, 2017, pp. 1–14.