ADDING MACHINE INTELLIGENCE TO HYBRID MEMORY MANAGEMENT

A Thesis Proposal
Presented to
The Academic Faculty

By

Thaleia Dimitra Doudali

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

December 15 2020

ADDING MACHINE INTELLIGENCE TO HYBRID MEMORY MANAGEMENT

Thesis committee:

Professor Ada Gavrilovska, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Vivek Sarkar
School of Computer Science
Georgia Institute of Technology

Professor Alexey Tumanov
School of Computer Science
Georgia Institute of Technology

Professor Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhanva Gurumurthi
Advanced Micro Devices Inc. (AMD)

Date approved: December 15, 2020

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The integration of non volatile memory technologies into the main memory substrate enables massive memory capacities at a reasonable cost in return for slower access speeds. Preserving the application performance levels of traditional homogeneous memories is feasible with fine-tuned data management approaches. At the same time, resource management solutions augmented with machine learning show great promise for fine-tuning system configuration knobs and predicting future behaviors. This thesis explores such trends, reveals new insights and closes the application performance gap left by existing solutions, with the following contributions.

First, this thesis builds Kleio; a hybrid memory page scheduler with machine intelligence. Kleio deploys Recurrent Neural Networks to learn memory access patterns and optimizes the selection of which data to dynamically move across the memory units. Kleio cleverly focuses the machine learning on the page subset whose timely movement will reveal most application performance improvement, while preserving history-based lightweight management for the rest of the pages. In this way, Kleio bridges on average 80% of the relative existing performance gap, while laying the grounds for practical machine intelligent data management with manageable learning overheads.

Second, this thesis contributes Cori; a system-level solution for tuning the operational frequency of periodic page schedulers for hybrid memories. Cori synthesizes information on data reuse to properly identify the data movement frequencies to be tested, reducing by 5$\times$ the number of tuning trials compared to existing empirical or insight-less tuning approaches. In this way, Cori delivers application performance levels within 3% from the case of optimally selected frequency, eliminating the 10%-100% performance gap created when using frequencies currently adopted by related works. Such improvements translate to substantial gains when considering the scale of emerging supercomputing and datacenter systems.

Finally, this thesis proposes a holistic machine intelligent page scheduling solution that brings together insights from Cori and the design principles of Kleio. The outcome is a page scheduler that leverages similarity in page access patterns to afford use of machine intelligent management on more pages than Kleio, while amortizing training overheads. Combining this with the use of a cost-efficient operational frequency, as identified by Cori, enables maximum performance improvements that otherwise cannot be practically realized with acceptable costs.

CHAPTER 1
INTRODUCTION

Heterogeneous hardware emerged to address the slowdown of Moore's Law and the exponentially growing demand for compute and storage resources by popular big data analytics, applications of artificial intelligence and scientific simulations. Regarding the memory substrate, non volatile memory technologies of massive capacity capabilities emerged to enable fast data retrieval and storage for data-intensive workloads, in response to the scaling limitations and skyrocketing cost of the volatile DRAM. For instance, Intel's Optane DC persistent memory [1] provides terabytes of memory at 1/3 of DRAM's cost [2]. Intel offsets the at least $3\times$ slower access speeds of persistent memory [3] by packaging together gigabytes of DRAM, which account for as little as 6% of the platform overall memory capacity, thus creating a hybrid memory environment.

The efficient resource management of hybrid memory, via proper data tiering, allows for the desired performance levels of the aforementioned classes of applications. To achieve this, well established approaches in hybrid memory management build the necessary mechanisms to maximize the utility of the fastest available memory component via corresponding dynamic movement of frequently accessed data. The task to identify which data is most appropriate to move and at what times, is particularly challenging, depending on the available data access information and performance estimates. Current solutions span the software stack from algorithm- [4, 5], profiling- [6, 7], library- [8], runtime- [9, 10, 11] to operating system-level [12, 13, 14, 15, 8, 16] solutions. Custom APIs [9, 6, 17, 8] and specialized hardware [13, 12, 18, 19, 15, 20] are frequently proposed as part of the solutions across the software stack, to collect the necessary data access information and deliver the desired performance levels.

However, the ever increasing complexity of emerging workloads and of the system parameter configuration space, breaks the effectiveness of current system-level and application-agnostic solutions. More specifically, the use of current heuristics, that are fine-tuned for conventional workloads, is detrimental for analytics with more intricate or random accesses. In addition, the effectiveness of heavily used empirical configurations is vulnerable to such emerging workloads, due to the impermissible overheads of fine-tuning all possible combinations of parameters and scenarios. Therefore, existing approaches are not robust across classes of emerging workloads and configurations of the heterogeneous hardware. This results in substantial loss in performance and resource efficiency of the heterogeneous memory hardware.

The increased complexity and resulting performance gap call for more intelligent and insight-based solutions compared to existing human-derived heuristics and settings. Yet, system-level solutions need to be lightweight so as to be commercially adopted and to maximize performance with minimal software-level overheads. Although the effectiveness of machine intelligence in addressing complexity is very appealing for the purpose of hybrid memory management, its system-level adoption needs to be very judicious, due to the non-trivial learning overheads. This raises a number of questions: How to preserve the practicality and responsiveness guarantees of such a system-level resource management solution? When is machine intelligence necessary to use? How can machine intelligence co-exist with human intelligence and compliment existing lightweight system-level mechanisms? We next discuss, in more detail, the need for machine intelligence and the challenges its system-level integration introduces.

1.1 Statement of Problem

- **Emerging complex workloads break the effectiveness of conventional approaches.** The growing adoption of machine learning methods across application domains creates a new class of workloads that require adequate system-level support [21, 22, 23, 24, 25, 26]. Similarly, scientific simulations now capture even more complex phenomena and relations [27]. These applications and methods have vastly more intricate execution phases and access patterns, than traditional analytics. Yet, hybrid memory management approaches have not evolved to address this newfound complexity in data access behaviors, and continue using heuristics and approaches that are only proven to work well across classes of conventional workloads.

In more detail, the use of access history information to predict future access behaviors has been effective for traditional classes of workloads with sequential, strided and regular access patterns. However, it generates vastly inaccurate predictions when reacting to sudden changes in execution phases or to completely irregular behavior, which is predominant in emerging workloads. This results in an inefficient selection of data to be moved across the memory components, due to mispredictions of upcoming patterns. In consequence, application performance degrades due to the suboptimal data tiering, as well as the wasteful resource utilization that delivers such data migrations. The created performance gap is substantial and requires novel access pattern prediction mechanisms enriched with machine intelligence to realize the full potential of the heterogeneous hardware.

- **The increased complexity in the parameter configuration space hinders the fine-tuning of critical parameters.** The large configuration space of heterogeneous hardware, the intricate resource requirements of emerging workloads and the explosion of solution-level parameters, vastly complicate the configuration space. This leads to the empirical setting of certain knobs, in an effort to minimize the associated tuning overheads. For instance, the frequency of data movements over hybrid memory has always been empirically set at certain fixed values, that surprisingly varies substantially across related works [12, 14, 28, 13, 15]. In consequence, such settings are not robust to new workloads classes or scenarios that were not included in their experimental evaluation. However, the importance of this parameter is enormous since its operation is on the critical path of hybrid memory management, where a misconfigured value can lead to tremendous aggregate movement overheads. An insight-based tuning of this important parameter that enables minimal tuning effort, will be more effective and practical than current insight-less settings, and can deliver the full benefits from the heterogeneous hardware.

- **The newfound complexity generates complicated, application-specific and impractical solutions.** To address the new requirements and behaviors of emerging workloads, researchers develop solutions with more intricate performance models [10, 29] and heavy profiling [6] that increase the management decision overhead, rely on new hardware support [13, 12, 29, 19], or resort in application-guided [6, 8] or runtime-specific [9, 10, 11] solutions. Yet, commercial system-level support for emerging hardware chooses practicality over robustness and effectiveness across applications. System-level solutions need to be lightweight and responsive to adhere to decision time guarantees and to seamlessly cooperate with other system-level components. For instance, to offer support for new technologies such as Intel Optane DC PMEM [1], Linux simply extended its well established and lightweight `autonuma` component, to enable identification and migration of hot and cold pages across DRAM and persistent memory [30]. Therefore, it is essential to preserve the practicality guarantee, while delivering more intelligent hybrid memory management. Although machine intelligence can alleviate the human effort in this complex decision space, its insight-less adaption in the decision pipeline comes with unacceptable learning overheads. This is due to the fact that the problem size of system-level hybrid memory data tiering is enormous, since it involves the placement of every singe page of applications with enormous memory footprints, that are target for hybrid memories. Therefore, it is critical that machine intelligence is used in a practical way, complimenting existing lightweight approaches, and only to the extent that is necessary to deliver the desired performance levels under permissible overheads.

1.2 Thesis Statement

To deliver practical solutions that maximize the performance benefits and the efficiency from emerging heterogeneous memory hardware, existing lightweight system-level solutions need to be augmented with judicious use of machine intelligence.

1.3 Contributions

In support of this statement, this thesis makes the following contributions.

- **Machine intelligent hybrid memory management.** To address the inefficiency of existing approaches to accurately predict the complex access behavior of emerging workloads, this thesis introduces machine intelligence into the hybrid memory management. It proposes Kleio; a hybrid memory page scheduler with machine intelligence [31]. Kleio deploys Recurrent Neural Networks (RNNs) to learn memory access patterns, that existing history-based solutions fail to accurately predict. Kleio does so at the granularity of individual pages, to learn the pattern of their access frequency across periods of time. The novelty in the design of Kleio comes from the selection of a small page subset, whose machine intelligent management reveals most of the application performance improvement, while using existing history-based management for majority of the pages. The resulting hybrid management approach lays the ground for Kleio's practical integration at the system-level. Kleio's impact is extremely promising, since it bridges on average 80% and up to 95% of the existing performance gap.

- **Insight-based hybrid memory management tuning.** To address the inefficiency and inconsistency of empirically configured data movement frequencies across related works and classes of application, this thesis builds a tuning solution based on insights that can be captured at the system level. In more detail, it proposes Cori;

a system-level solution for tuning the operational frequency of periodic page schedulers for hybrid memories [32]. The novelty of Cori derives from observations on data reuse times and their alignment with the data movement frequency. Cori synthesizes information on data reuse to properly identify the data movement frequencies to be tested, reducing by $5\times$ the number of tuning trials compared to existing empirical or insight-less tuning approaches, and realizing almost maximum possible application performance levels, within only a trivial margin of 3% on average from the case of optimally selected frequency. Thus, Cori enables the proper fine-tuning that enables maximum performance improvements that are suppplementary to the ones deriving from machine intelligent management.

- **Reduction of machine intelligent hybrid memory management overheads.** To address the complex decision process of hybrid memory management, especially when it relies on machine intelligence, this thesis proposes a mechanism to further reduce the associated learning overheads. Although, this thesis builds upon judicious use of machine intelligence portrayed in the design principles of Kleio, the associated overheads can vary $3\times$ - $4\times$ across applications, depending on their data input sizes and irregularity in access behavior. In response, this thesis builds a page grouping mechanism to further reduce the number of individual Recurrent Neural Networks (RNNs) deployed by Kleio, as previously described. The novelty of this work derives from utilizing insights on data reuse times, as highlighted in Cori [32], to identify large page groups with the same access patterns. The resulting fusion of Kleio's design principles and Cori's data reuse insights leads to a holistic page scheduler, Kleoniki, which delivers maximum performance improvements via the fine-tuned and machine intelligent hybrid memory management. Kleoniki, enables the machine intelligent management of $2\times$ more pages than Kleio [31], under the same training overheads, making it possible to extend the benefits of Kleio to more complex and data intensive workloads in a practical manner.

CHAPTER 2
BACKGROUND

This chapter provides background information on emerging heterogeneous hardware technologies, their commercial availability and deployment in datacenter and exascale computing environments. In addition, it highlights the increased complexity in the configuration of the hybrid memory and the data access behaviors of emerging workloads. Finally, this chapter summarizes the existing solutions in the hybrid memory management space and identifies their limited effectiveness against the newfound complexity.

2.1 Complexity in Hybrid Memory Management

The increasing complexity in hybrid memory management derives from the extreme heterogeneity of emerging memory hardware technologies and their configuration, together with new resource requirements and access behaviors of emerging application classes.

2.1.1 Emerging Hybrid Memory Platforms

In the current post-Moore era of computing, the traditional model of homogeneous DRAM-only memory systems is replaced with heterogeneous hardware to massively scale the main memory capacity under permissible system cost. Figure 2.1 summarizes the recent trends in hybrid memory configurations that include emerging hardware technologies and resource disaggregation techniques. New hardware technologies introduce different trade-offs of cost, speed, capacity and capabilities in programmability and data persistence. For instance, Intel's Optane DC Persistent Memory [1] (PMEM) provides terabytes of persistent data storage, at around $3\times$ times cheaper than DRAM [2], in return for at least $3\times$ slower access speed [3, 33]. Yet, with the appropriate data management the 375 gigabytes of available DRAM, packaged together with 6 terabytes of PMEM, are sufficient to deliver

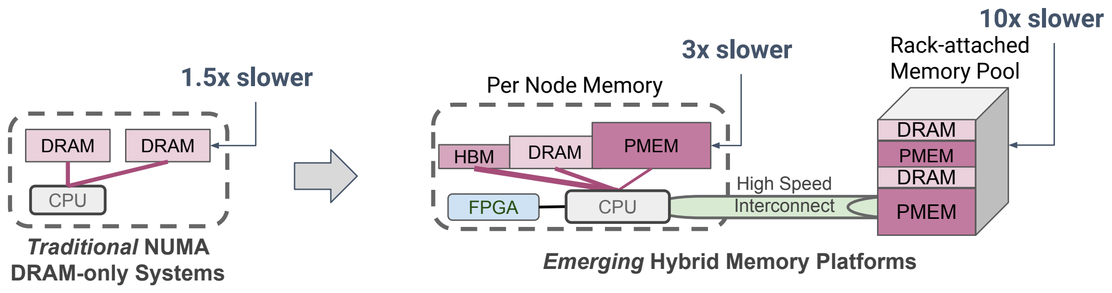

Figure 2.1: Heterogeneous memory hardware and memory disaggregation provide massive memory capacities in return for bigger disparity in the access speeds and configuration of the memory substrate, compared to traditional homogeneous systems.

DRAM-like levels of application performance [3]. Similarly, High Bandwidth Memory (HBM) deliver 5× - 10× more bandwidth compared to DRAM technologies [34]. It is widely used in the TOP500 Supercomputers at capacity of tens of gigabytes per node, either standalone [35] or together with DRAM [36]. Finally, well established techniques of resource disaggregation [37, 38], now aggregate both volatile and non volatile node-local memory resources into a massive shared pool of remotely accessible hybrid memory, and deliver high-speed data transfers with novel interconnection fabrics [39, 40].

2.1.2 Configuration and Tiering Complexity

The increased number of distinct memory technologies in the main memory substrate adds new dimensions in the complexity of their configuration. Each hybrid memory unit may exhibit different properties in terms of speed, capacity, programmability or other parameters, complicating the decision regarding their optimal configuration. In addition, the available memory components can be organized in two primary ways or in a combination thereof. As depicted in Figure 2.2, for the example case of two memory tiers, there is the vertical and horizontal hybrid memory organization. In the vertical (otherwise cache) organization, one memory unit acts as a cache for the other and is managed by the hardware. In the horizontal (otherwise flat) organization, all memories 'lay flat' and are managed by software – the operating system or applications themselves. For instance, these correspond to the

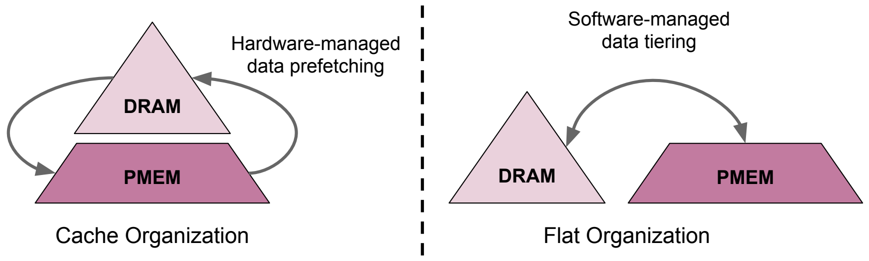
Hardware-managed data prefetching
DRAM
PMEM
Cache Organization
Software-managed data tiering
DRAM
PMEM
Flat Organization

Figure 2.2: The cache organization of hybrid memory enables hardware-managed data prefetching techniques from PMEM to DRAM. The flat organization of hybrid memory allows for software-based control of the data tiering across DRAM and PMEM.

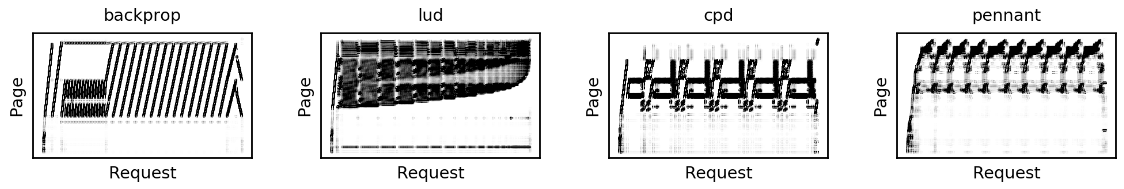

Figure 2.3: Memory access patterns of emerging workloads with increasing complexity in data access behavior. From sequential strides (`backprop`), triangular traversals (`lud`), sparse tensors (`cpd`) to iterations of random accesses (`pennant`).

Memory and *App-direct* modes in Intel's Optane DC PMEM platform [1]. Each organizational mode introduces different trade-offs with respect to system resource efficiency and application performance. For instance, recent work has shown that the cache organization improves performance of graph applications [41]. In contrast, the flat organization allows for lower energy cost and higher bandwidth use [42, 33], and a number of hardware and software techniques have recently been proposed to further improve the associated management overheads [13, 12, 18, 19, 15].

2.1.3 Emerging Complex Workloads

New classes of popular workloads include machine learning methods with complex matrix operations, massive graphs of random connectivity and scientific simulations that capture irregular behaviors and phenomena. Figure 2.3 captures examples of such access patterns in increasing order of complexity. New or extended benchmark suites [21, 22, 23, 24, 27, 43] introduce such emerging workloads and together with well established suites [25, 26, 44, 45] drastically augment the application classes that require robust support for high performance over heterogeneous hardware in particular.

Emerging workloads add complexity to the hybrid memory management because it is harder to predict the randomness, frequent phase changes and other irregularities in their access behaviors, compared to conventional workloads. The effectiveness of hybrid memory tiering relies on accurately forseeing the upcoming trends in access patterns, so as to timely migrate future frequently accessed data in the fastest memory unit. This is particularly challenging given the use of past access history, that is readily available, since it is not always sufficient to predict new patterns or old trends outside the retained information. This results in a selection of data movements that do not optimize the data tiering, reduce the utility of the fastest memory and waste resources with non useful migrations. This translates in degradation of performance and efficiency. Therefore, such emerging classes of applications require hybrid memory management approaches with more predictive capabilities and accurate access pattern predictions.

2.2 Related Work

This section summarizes hybrid memory management solutions across the software hardware stack, evaluated both on prior simulation and emulation environments of emerging hardware, such as persistent memory, as well as on recent commercially available hardware platforms.

2.2.1 Software Solutions

Starting at the top of the software stack, inside applications themselves, related works optimize the algorithmic design to perform more efficiently over the underlying hardware. For instance, the k-means NUMA Optimized Routine (knor) library [4] optimizes k-means for modern NUMA architectures and minimizes synchronization barriers. Similarly, algorithm features, common numerical operations, and algorithm structures can be used to direct data placement for conjugate gradient, fast Fourier transform, and LU decomposition of a matrix [5]. In addition, application-level hints are widely used by related works across the software stack to appropriately guide the focus of the hybrid memory management into user-identified critical data structures and regions. To this extent various solutions propose custom data allocation APIs, that require application source code modifications, to improve initial and dynamic data placement of the marked data regions. [6, 7, 9, 17, 11].

Regarding contributions at the user library-level, Memkind [8] is a user extensible heap manager that can be adopted by other middleware solutions to improve performance over heterogeneous memories. Similarly, runtime solutions allow for easy detection of execution phases to properly align data monitoring and movement time intervals. More specifically, Unimem [9] leverages the MPI communication phases to launch data movements and Tahoe [10] aligns task-based execution with corresponding data migrations. Finally, at the middleware-level various solutions rely on application profiling of data access behaviors, build performance and data movement cost models to optimize data tiering [6, 7, 11].

Lastly, operating system-level solutions that are application-agnostic rely on page access information available on the kernel's page tables, to identify frequently accessed pages and to periodically migrate them. Certain solutions leverage existing NUMA-based page migration support [12, 14, 15, 20, 28], or appropriately extend NUMA-based data balancing policies [16, 46]. Spanning across the software stack, Memif [17] introduces a user interface, a user space library and kernel space service to accelerate page migrations across hybrid memory.

The complete software-level solution space also includes optimizations in the initial static data placement across hybrid memories. Our own prior work improves the hybrid memory resource efficiency upon shared use [47] and maximizes the system cost efficiency for datacenter oriented workload classes [48] via optimized static data tiering. Additionally, CHOPT [49] aims to be an optimal offline algorithm for data placement over multi-tiered heterogeneous systems. Finally, the effectiveness of various static data placement methods has been evaluated against disaggregated memory systems with non volatile memories [50].

2.2.2 Hardware Solutions

Software-level solutions create significant associated overheads, which often are impractical for the decision time guarantees of resource management solutions. To this extent, a significant body of the aforementioned software-level solutions propose *specialized hardware* to reduce critical overheads. For instance, hardware-assisted page hotness tracking with custom hardware elements is highly suggested by operating system-level solutions [13, 12, 18, 19, 15, 20]. The availability of such hardware is critical in providing similar performance levels between the different organization modes of tiered memory, that is software-managed flat memory mode and hardware-managed cache memory mode.

Hardware-based solutions introduce custom counters to monitor data accesses and enable threshold-based data migration triggers [51, 29]. In addition, additional hardware buffers for data copies enable access to data that is under migration, reducing associated stalls and improving application performance [29]. Custom memory controller hardware is also proposed to enable support for page migrations in disaggregated non-volatile memories [18]. In addition, Mempod [19] builds a clustered architecture that enables scalable migration support over multi-level memories, that outperforms prior work which transparently manages hybrid memories configured in a combination of cache and flar organization [52, 53].

2.2.3 Solutions with Machine Intelligence

The adaptation of machine intelligence as part of the hybrid memory management decision process is currently very limited across related works. For instance, the runtime solution Tahoe [10], which targets task-based program execution, incorporates machine intelligence to improve the accuracy of their performance modeling. More specifically, it deploys an artificial neural network (ANN) of limited size to predict task execution times, given the complex relationship of input features on last level cache miss rates and instructions per cycle. However, for more complex placement scenarios the authors use an analytical model, since training and infering upon a more complex machine learning model comes with impractical overheads.

Yet, the effectiveness of machine intelligence in other similar systems problems shows great premise. For instance, LLAMA [54] is a memory allocator with machine intelligence that aims to reduce the memory fragmentation of huge pages on C++ server workloads. It deploys recurrent neural networks to learn and predict data object lifetimes. Furthermore, cache prefetching techniques have higher precision and recall with offline machine learning of data access patterns [55]. Next, Selecta [56] utilizes latent factor collaborative filtering, in order to find the configuration of cloud compute and storage resources that provides optimal cost-to-performance trade-offs. Finally, machine learned index structures [57] and replacing parts of database management stack with machine intelligence based components provide significant performance benefits.

2.3 Summary

Related work so far has primarily focused on providing the necessary support to enable effective data tiering across multi-tiered hybrid memory. This is done via readily available operating system-level information, or guided by information provided by the application, runtime, or detailed access profiles, often times together with the appropriate hardware support. However, the current approaches and heuristics adapted are not effective towards the increased complexity in data access behaviors and heterogeneous hardware configurations, as this thesis identifies. Although machine intelligence shows great premise in similar resource management problems, its adaptation in hybrid memory management so far is limited due to the impractical learning overheads of an insight-less adaptation. Yet, the judicious use of machine intelligence in collaboration with lightweight existing methods, is proven to enable permissible latency and prediction accuracy in similar problems. Therefore, this thesis contributes the design principles for the practical integration of machine intelligence in hybrid memory management.

CHAPTER 3
COMPLETED WORK

This chapter includes information on the methodology of the experimental evaluation and terminology used throughout the thesis. Then, for the purpose of the thesis proposal, this chapter summarizes the key contributions of its completed work.

3.1 Methodology

Applications. The experimental evaluation is performed against a collection of applications, benchmarks and kernels from well established as well as newer benchmark suites, which are PARSEC [25], Rodinia [26], CORAL [27], ParTI! [23] and Lonestar [24]. The selected benchmarks and mini-apps cover a wide range of application domains and regular, irregular and random memory access patterns.

Hybrid Memory Simulation. To evaluate upon a broad spectrum of hybrid memory platforms and enable fine-grained control on its configurations, this thesis builds a lightweight simulation infrastructure. In more detail, it is a Python-based simulation environment[1] that allows fast trace-based analysis similar to [13]. In particular, it assumes a flat organization of fast (e.g., DRAM) and slow (e.g. PMEM) memory, similar to the App Direct mode configuration of the Intel Optane persistent memory platform [1]. Following the observed PMEM access speeds [3] the latency is set at a 1:3 ratio and the bandwidth at a 1:0.37 ratio between the fast and slow memory. The overall capacity of the memory system is equal to the application's memory footprint. Since it is not a cycle-accurate simulation, it assumes that a period is the time duration when a fixed number of memory requests are issued, e.g., 1,000 requests per period. The runtime is estimated as the aggregate access latency of the memory requests for their coresponding memory allocation across periods. In addition, the

[1]https://github.com/GTkernel/cori-sim.git

case of limited bandwidth availability is accounted by injecting appropriate delays given the number of memory requests serviced over a window of time. Finally, constant delays are added for every page migration and start of a period to account for the overhead of the page scheduler itself, using the proposed values in [13, 18].

Memory Access Trace Collection. Input to the aforementioned hybrid memory simulation are application profiles in the form of memory access traces. These are collected using Intel's Pin [58] dynamic binary instrumentation tool. In addition, externally collected traces were also provided for the evaluation of this thesis. These are collected using the Instruction Based Sampling (IBS) mechanism that is available on AMD's processors [59]. Also, other such external traces were collected by intercepting the last level cache misses on an actual hardware platform with an AMD A10-5800K APU clocked at 3.8GHz.

The memory access traces capture the address of last level cache misses, whether it was a load or store operation, the corresponding program counter and in certain cases a timestamp. In order to allow for reasonable trace sizes and analysis times the traces collected with Intel's Pin tool are generated from a cache hierarchy of smaller but proportional capacity ratio to the Intel Optane DC PMEM platform. The application data inputs are proportionally fixed such that they receive similar last level cache miss rates to application execution in the native PMEM platform.

Page Scheduler. In the context of this thesis, the main system-level component in hybrid memory management is referred to as a page scheduler. It periodically collects information of page access counts and migrates pages between fast and slow memory, according to the scheduling policy. To this extent, this thesis extends the Python-based simulation with the implementation of a page scheduler that periodically aggregates per page access counts from the collected access trace, that is given as input. The initial page allocation is done in an interleaved manner across memories, which is typical for NUMA systems. Then, upon every period the implemented scheduling policy indicates the migration of frequently accessed (hot) pages that reside in slow memory to faster memory, replacing any recently used (LRU) pages. The number of page migrations per period is capped by the available fast memory capacity, since hot and LRU pages are swapped across hybrid memory. These page swaps happen asynchronously, assuming DMA support, and sequentially in order of (hot, LRU) page pairs.

This type of page scheduler, that makes a selection of page migrations using access history, is referred to as a **history** or **reactive** page scheduler, since it 'reacts' to the changes in the memory access patterns, as done in existing solutions thus far [12, 14, 28, 13, 15]. The simulation also includes an **oracle**, otherwise named as **predictive** page scheduler, that predicts memory access patterns, thus makes a more sophisticated page migration selection or even has a-priori knowledge of the access pattern, as described as the oracular baseline in [13]. This simulation configures the history/reactive page scheduler to act upon a single period of past access history, and similarly the oracle/predictive page scheduler to make an access pattern prediction for the upcoming period.

Validation of the Simulation Infrastructure. The accuracy of the aforementioned simulation environment is validated on a server with Intel Optane DC Persistent Memory (PMEM), configured in *App Direct* mode. Application performance analysis in the simulation environment with the memory access traces captured with Intel's Pin tool, delivers similar performance levels as in the PMEM platform, validating its accuracy. The machine contains 375 GB of DRAM and 6 TB of PMEM. A custom page migration module[2] for Linux kernel version 5.4 attaches to a target process and periodically selects 4 KB pages to move between DRAM and PMEM. Every period, it identifies page accesses using the available OS-level information, as also done in [15]. The module determines which pages were accessed by scanning the target's page table entries and recording whether or not each accessed bit was set during that period. To estimate the page hotness, it calculates the exponential moving average (with a certain smoothing factor) of the page's accessed bit history and compare it with a hotness threshold that classifies a page as hot or cold, as also done

[2]https://github.com/GTkernel/x86-Linux-Page-Scheduler.git

in [28]. Then, utilizing the `move_pages()` function from the kernel's NUMA-based migration API, it asynchronously moves hot pages to DRAM and cold pages to PMEM. The kernel module dynamically adjusts the page migration cutoff, dividing the process memory footprint across DRAM and PMEM at a certain capacity ratio.

3.2 Kleio: Machine Intelligent Hybrid Memory Management

3.2.1 Motivation

Well established operating system-level approaches for data tiering over hybrid memories, include a *page scheduler* component, that periodically monitors and aggregates page access counts, so as to move frequently accessed (hot) pages in DRAM replacing cold ones. The effectiveness of this approach relies on the accuracy of the page hotness prediction. Existing solutions use information over a short window of past access behavior, to predict which pages will be hot in the future, so as to timely move them in DRAM. Yet, the use of purely historical information is not robust towards sudden changes or randomness in page access patterns. In such cases, this translates to a poor selection of page migrations by the page scheduler that wastes resources, reduces the efficiency of data tiering and ultimately hurts application performance.

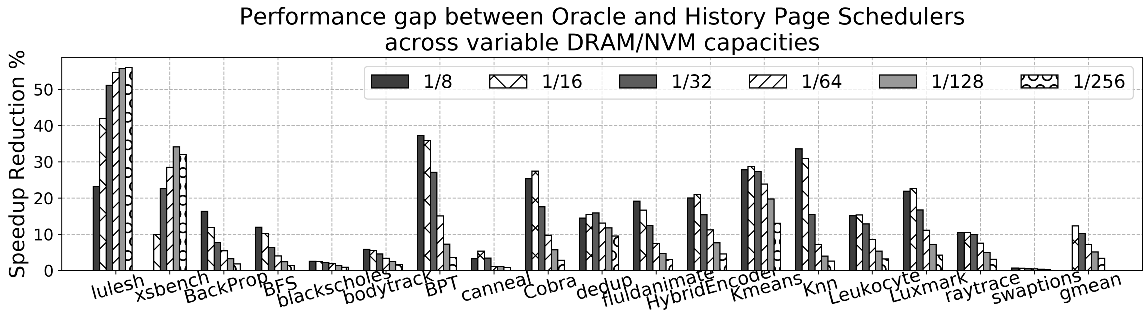

Figure 3.1: Performance gap due to sub-optimal data movement selection as a result of history-based access pattern predictions.

Figure 3.1 shows that existing history page schedulers, that use purely history-based data access information, make sub-optimal data movement selections, resulting in up to

50% performance degradation, compared to oracle page schedulers with a priori knowledge of the access patterns. Therefore, there is an opportunity to realize the full benefits of the heterogeneous memory hardware via more accurate predictions of data access behavior.

3.2.2 Contribution

To bridge the significant performance gap, this thesis leverages machine intelligence to enable more accurate predictions, thus optimized data movement selection. In particular, it builds Kleio[3]; a hybrid memory page scheduler with machine intelligence [31]. Kleio deploys Recurrent Neural Networks (RNNs) to learn complex page access patterns. The primary design principle of Kleio is to enable the practical integration of machine intelligence as part of the system-level hybrid memory management decision space. This is extremely challenging due to the complexity of machine learning methods, their non trivial learning overheads and the massive input space of page access behavior across target workloads of exascale and datacenter environments. Therefore, an insight-less application of machine intelligence across all pages leads to impermissible learning overheads.

To enable a practical solution, Kleio adopts a hybrid page management approach based on the following insights. First, not all pages *need* machine intelligent management. Existing lightweight history-based solutions, are able to produce accurate predictions for certain regular patterns and execution phases. Therefore, the pages that require more intelligent management are the ones frequently misplaced by the history-based approaches. Second, observations show that across workloads there is a small subset, as low as 20% of the misplaced pages in certain cases, that gets accessed more frequently than the rest of the pages. Therefore, the machine intelligent management of this page group, its accurate pattern prediction and optimal dynamic data placement can deliver majority of maximum possible performance improvements.

Based on these insights, Kleio's design is summarized in Figure 3.2. Kleio features a

[3]The name is inspired by ancient Greek mythology, where Kleio is the muse of history, daughter of Mnemosyne, goddess of memory

Page Selector component that identifies the page subset that will be managed by machine intelligent access pattern predictions. Kleio deploys individual per page RNNs to learn offline the pattern of page hotness across scheduling periods. During the online application execution, upon every period, the page scheduler executes model inference on the respective trained models and uses history-based information on the rest of pages, to predict their page hotness. Then, the page scheduler moves frequently accessed pages to DRAM, replacing cold ones to PMEM, depending on the data tiering at the time and the respective memory capacities.

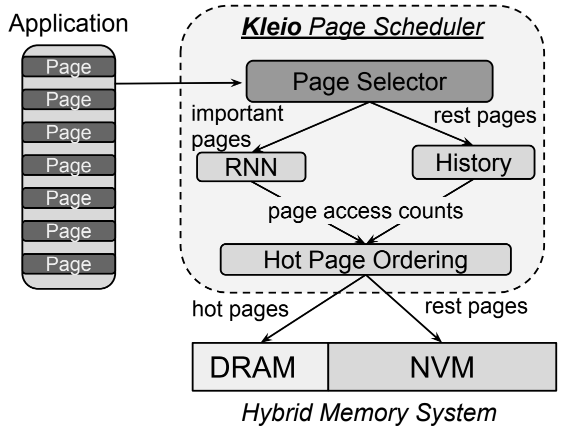

Figure 3.2: System design of Kleio, a hybrid memory page scheduler with machine intelligence.

3.2.3 Evaluation Highlights

Figure 3.3 shows the application performance that Kleio achieves. We observe that in most cases, the RNN predictions are sufficiently accurate to bring **80%** of the possible performance improvement, on average and more than **95%** for half of the applications that we considered, almost eliminating in full extent the aforementioned performance gap. Unfortunately, there are cases such as `bodytrack` and `raytrace`, where less than 50% of the possible speedup is achieved, in which case more pages need to be trained so as to further provide significant speedup.

In conclusion, the use of machine intelligence in hybrid memory management solves

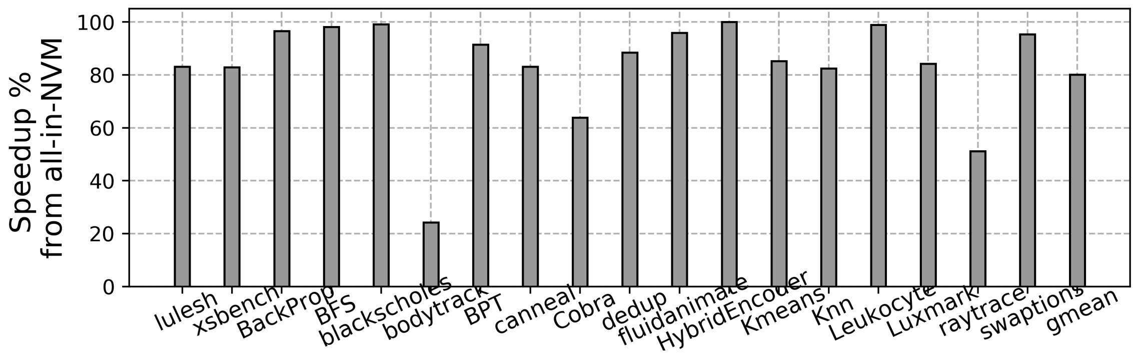

Figure 3.3: Application performance achieved with Kleio, normalized between 0, that is the case where all pages are managed with purely history-based information and 100, that is the case of oracular page management.

the inefficiency of current approaches against the increasing complexity of memory access patterns and reaches performance levels otherwise attainable under oracular management. Kleio's hybrid design, that leverages existing lightweight approaches and applies machine intelligent to cleverly selected pages, lays the ground for the practical integration of machine intelligent memory management solutions in future systems.

3.3 Cori: Insight-based Hybrid Memory Management Tuning

3.3.1 Motivation

The growing size of the configuration space of heterogeneous hardware and the substantial overheads of properly setting each parameter, leads to *empirical* tuning of the operational frequency of periodic hybrid memory management solutions. Suprisingly, the values selected by related works, as summarized in Table 3.1, vary within orders of magnitude, hinting towards potential ineffectiveness of these values for application classes and configurations not included in their tuning process. Indeed, experimental evaluation of the effectiveness of these proposed values shown in Figure 3.4 reveals 10% - 80% performance degradation compared to the performance achievable with a best-case frequency, on average, across applications and page schedulers. In particular, no single proposed value works best across all applications and page schedulers. Therefore, there is an opportunity to

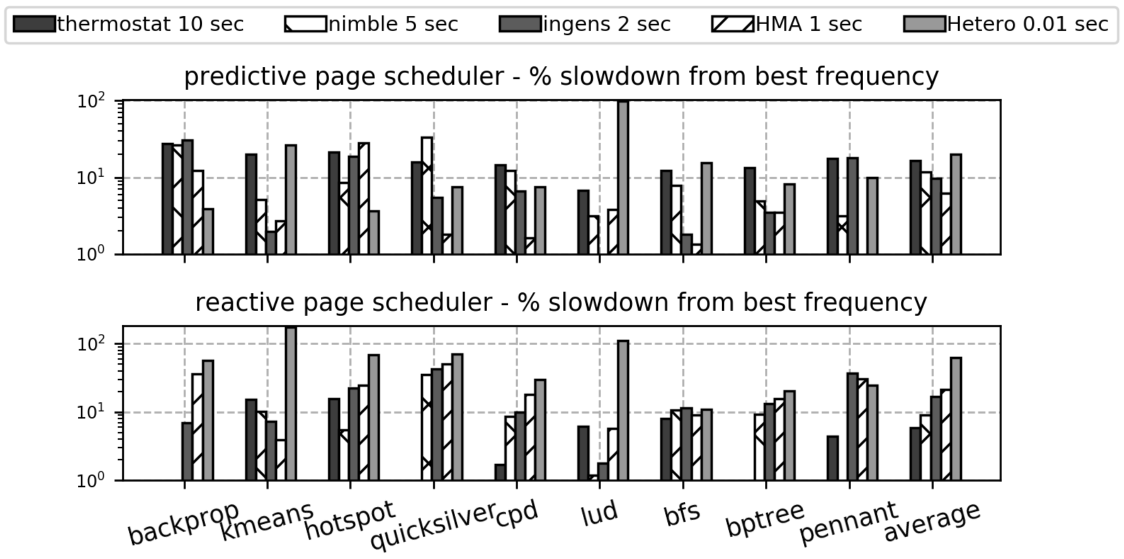

Figure 3.4: Performance gap due to sub-optimal data movement frequency selection as a result of empirical tuning of predictive and reactive (history-based) page schedulers.

close this performance gap with an insight-based tuning approach, rather than insight-less empirical procedures.

Table 3.1: Operational frequency of existing data tiering solutions

Solution	**Period Duration**
Thermostat [12]	10 sec
Nimble [14]	5 sec
Ingens [28]	2 sec
HMA [13]	1 sec
Hetero-OS [15]	0.1 sec

3.3.2 Contribution

To bridge this performance gap and avoid impermissible overheads of exhaustive fine tuning efforts, this thesis identifies data reuse insights to enable a sophisticated and limited selection of data movement frequencies to be evaluated. More specifically, this thesis reveals a relationship among the data reuse times and the page scheduling period durations which translate to best application performance. Page schedulers that use purely history-based access information benefit from periods that are larger than application data reuse

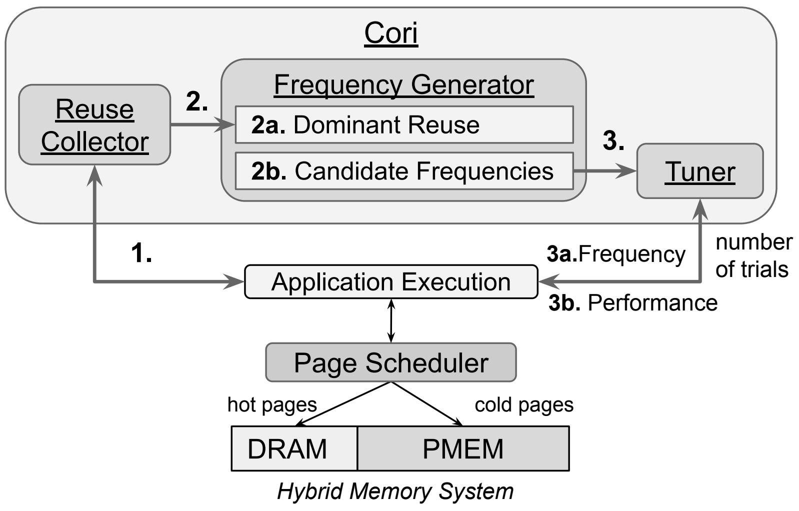

Figure 3.5: System design of Cori, a lightweight and insight-based solution for tuning the operational frequency of hybrid memory periodic page schedulers.

times, to make better page migration decisions. In addition, page schedulers should avoid very short periods that reveal the data monitoring and movement costs, as well as arbitrarily long periods that do not allow a prompt response to changes in the data access pattern and create insufficient aggregate data movement.

Based on these insights, this thesis contributes Cori[4], a lightweight and insight-based system-level solution for tuning the operational frequency of hybrid memory periodic page schedulers. Figure 3.5 depicts the system components of Cori, whose functionality is summarized as follows. First, the Reuse Collector executes a single profile run of the application to collect information on data reuse. Next, the Frequency Generator analyzes the data reuse profile and generates a range of proposed data movement frequencies. To achieve this, it first calculates the dominant reuse period as a weighted average of the observed reuses. Then, it generates a range of candidate frequencies at time intervals that are multiples of the dominant reuse period, and outputs the frequencies to the Tuner in decreasing order, from higher to lower frequencies, thus shorter to longer periods. Finally, the Tuner makes a small number of tuning trials with the candidate frequencies in the proposed order.

[4]The name is inspired by the ancient Greek mythology, where Cori (short for Terpsichore) was the muse of dance and daughter of Mnemosyne, the goddess of memory.

It configures the page scheduler to operate at each of the recommended frequencies. It then observes the application runtime and resource use and determines whether the application performance has reached best or desired levels. If not, the Tuner moves on to the next frequency in order, repeating the aforementioned trial process.

3.3.3 Evaluation Highlights

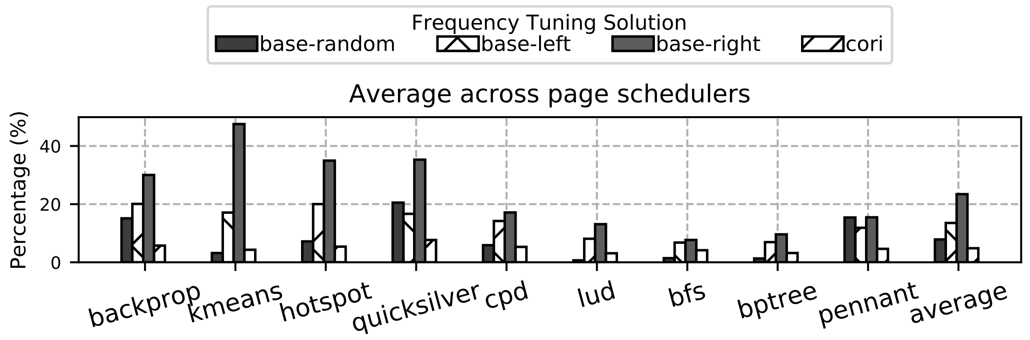

Figure 3.6: Performance slowdown from the case of optimal frequency selection. Cori provides performance only 3% away from the optimal one across applications and page schedulers.

Given the lack of a non-empirical tuning approach, Cori is evaluated against a custom baseline, which like Cori, operates at the system-level, but is blind to any insights it might have regarding application requirements. This baseline explores all possible frequencies which differ by a constant time step, which can substantially differ from Cori's dominant reuse period calculation. In particular, the `base-left` baseline starts from low frequencies (large periods) and moves to the left towards higher frequencies (short periods). The `base-right` baseline starts from high frequencies and moves towards the right to lower ones, similar to Cori. Third, we also assume a `base-random` approach that randomly explores all values.

Figure 3.6 includes the application performance that Cori enables compared to the insight-less baseline approaches, when executing for the same number of tuning trials that Cori requires to find best performance. The values are averaged across the page schedulers. Cori delivers lowest performance slowdown, only **3%** away from the optimal one,

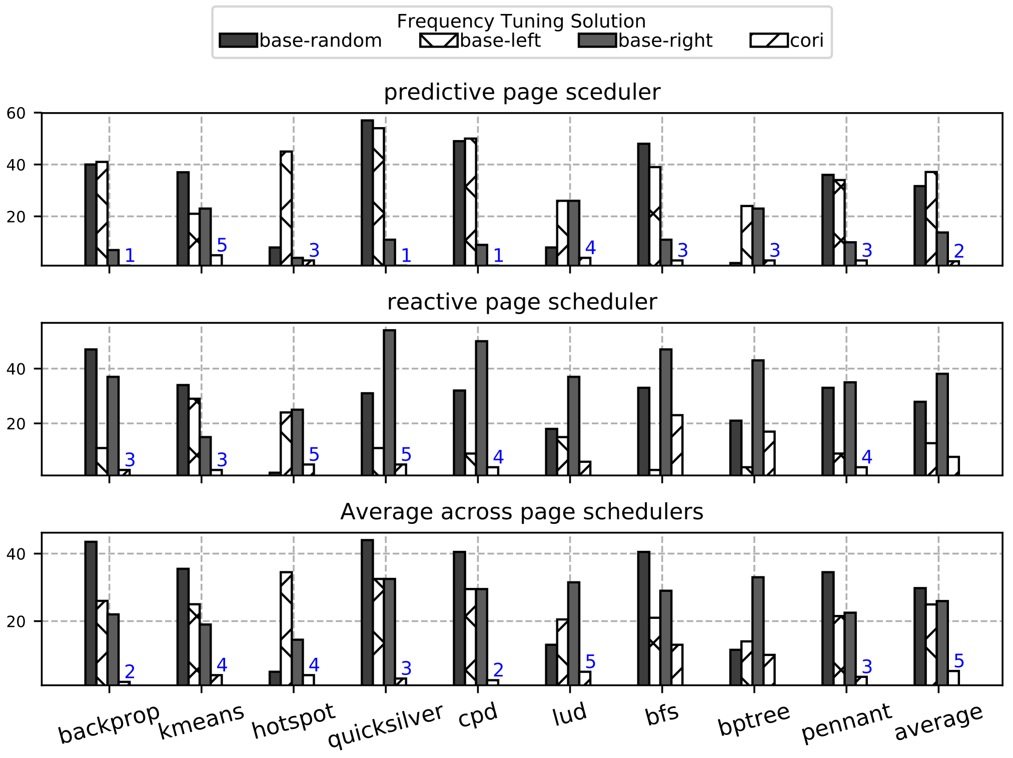

Figure 3.7: Number of tuning trials to find best performance. Cori (blue text) requires the minimum number of trials on average across all applications and page schedulers.

on average across applications and page schedulers. In contrast, the baselines incur higher performance slowdown because they require significantly more trials to reach best performance, as shown in Figure Figure 3.7. Within the execution overhead of Cori, only the `base-random` approach seems to be able to still choose frequencies that provide good performance, but only for some of the applications; for others (e.g., `quicksilver` and `pennant`), `base-random` is less effective even compared to some of the other baselines.

Regarding the tuning overheads of Cori, Figure 3.7 depicts the number of tuning trials it takes to find the highest application performance, that is only 3% away from the best possible one, as previously mentioned. Across applications and page schedulers Cori reduces the number of trials by $5\times$, from 25 on average across baselines down to only 5 trials. The only corner case where Cori requires up to 20 trials is for applications with random access patterns like `bfs` and `bptree` where the access pattern prediction capabilities of a reactive page scheduler are limited.

In conclusion, the proper frequency tuning of hybrid memory management solutions is critical to deliver the maximum levels of application performance attainable by the capabilities of the heterogeneous hardware. Cori's insight-based selection of data movement frequencies drastically reduces the number of tuning trials required to enable performance closest to the one possible via optimal frequency selection.

3.4 Summary

The completed thesis contributions introduce novel machine intelligent and insight-based elements in current hybrid memory management solutions, that eliminate existing performance gaps. The first contribution, Kleio, manifests the design principles that are necessary to lay the grounds for the practical integration of machine intelligence in system-level resource management. It does so by enhriching existing lightweight approaches, with judicious deployment of machine learning models over a limited number of pages, which are not efficiently managed by conventional methods. The second contribution, Cori, introduces data reuse insights in the tuning of the operational frequency of existing hybrid memory management solutions. Cori's lightweight fine-tuning delivers performance levels attainable with optimal frequency selection. The upcoming proposed work will bring together Kleio and Cori into a holistic page scheduling approach that delivers maximum performance improvements and resource efficiency, due to the combination of machine intelligence and data reuse insights.

CHAPTER 4
PROPOSED RESEARCH

This chapter makes a case for the proposed remainder of this thesis. It provides motivational numbers, an initial design and preliminary evaluation results of the proposed thesis contribution.

4.1 Motivation

The practical integration of machine intelligence into system-level hybrid memory management requires designs that minimize the associated learning overheads. This thesis contributes the necessary design principles, that leverage existing lightweight approaches to their fullest abilities and use machine intelligence judiciously. More specifically, Kleio deploys machine learning models to learn page access patterns over a small subset of pages, whose timely movement will maximize aggregate performance improvements. Yet, the absolute number of these pages depends on the overall application's memory footprint and the complexity in its data access behavior. More specifically, Figure 4.1 captures the application performance improvements as more pages are managed with machine intelligence, as prioritized by Kleio's Page Selector component. Workloads like `backprop`, `quicksilver`, `cpd`, and `bptree`, benefit tremendously from machine learning based management, and achieve 20% - 60% performance improvements compared to purely history-based existing solutions. For Kleio to realize these improvements the absolute number of per page neural network deployments would need to increase, and would vary $3\times$ - $4\times$ across these workloads. Depending on the actual memory footprint and input sizes of the application execution, these may lead to prohibitive learning overheads, given the available resources for machine learning. Therefore, it is crucial to further minimize the aggregate number of per page learning models, without sacrificing upon the attainable

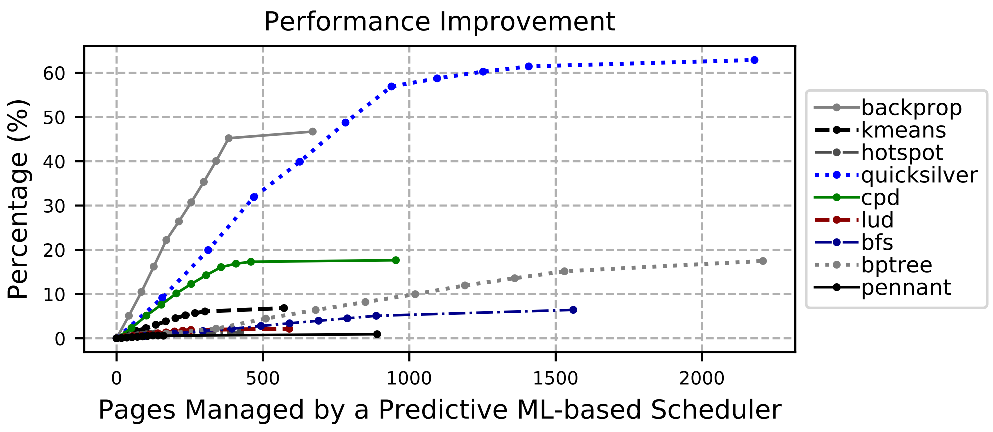

Figure 4.1: Application performance improvement across increasing number of pages managed under a more predictive machine intelligent page scheduler, as prioritized by Kleio's Page Selector component.

performance improvements.

4.2 Insight

To further minimize the aggregate learning overheads of machine intelligent data tiering, this thesis proposes a page grouping mechanism. More specifically, pages that are appropriate candidates for machine intelligent management via Kleio's sophisticated page selection, and which share similar access behavior across application runtime, are grouped together. The larger the groups the smaller the number of distinct learning models to train, thus reduced aggregate overheads. The training of a single model per page group allows a single model inference across the pages, which lies on the critical path of the hybrid memory management decision. The challenge in this approach is twofold. First, how to identify pages with similar access behavior and how is similarity captured and defined? Second, is a single ML model able to make accurate predictions across pages with similar patterns and if not, do we need to revisit the layout and hyperparameter tuning of the neural networks?

As a first step towards addressing these challenges and scoping the benefits from such an approach, the grouping mechanism identifies pages with *identical* access patterns. This

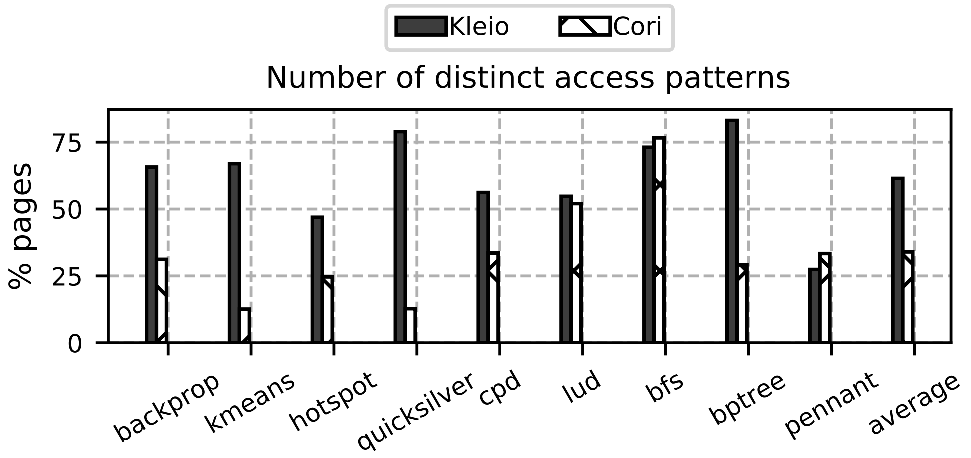

Figure 4.2: Number of distinct patterns of per period page access counts for Kleio's and Cori's selection of page scheduling frequency. Cori's sophisticated selection drastically reduces, on average, the distinct access patterns that potentially need more intelligent predictions.

eliminates any sensitivity of the ML model, since the inference across exactly the same input pages will incur the same accuracy levels. The ML models learn the pattern of the page access counts across *time*, that is the scheduling periods, therefore the duration of these periods impacts the sequence of values of the page access patterns. Revisiting Cori's data reuse insights and optimized page scheduling frequency selection, Figure 4.2 shows the number of distinct patterns of page access counts across applications for the selection of period duration that Kleio and Cori make. Cori makes a more coarse-grain period length selection than Kleio, that hides any minor differences in the page access counts, drastically reducing the number of distinct patterns across the pages of an application. For workloads like `kmeans` and `quicksilver` this reduction is as low as 20% of the application's overall memory footprint. A proper grouping of similar, not only identical, patterns and ML model tuning will reveal even larger benefits. Yet, this will be still challenging to achieve for workloads known for their random access behavior such as `bfs`.

Given these insights and potential of creating large page groups, this thesis proposes

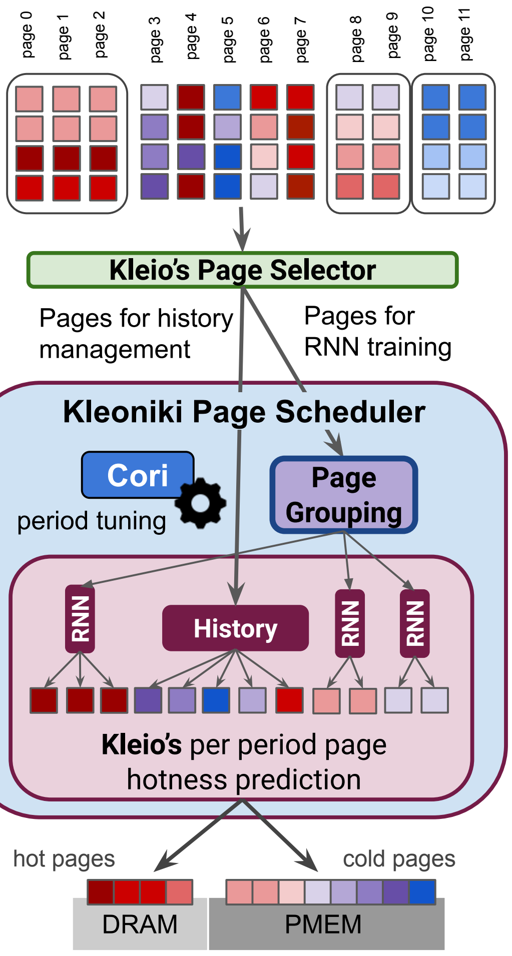

Figure 4.3: System design of Kleoniki, a holistic hybrid memory page scheduler with machine intelligence, fine-tuned operation and minimal learning overheads.

Kleoniki[1]; a 'victorious' extension of Kleio, that maximizes performance improvements via fine-tuned scheduling periods and minimizes the associated learning overheads. Kleoniki operates at the operational frequency identified by Cori and introduces a page similarity grouping mechanism after Kleio's Page Selector component, as shown in Figure 4.3. The mechanism clusters together pages with similar access counts across the scheduling periods. Then, individual recurrent neural networks are deployed to learn the access patterns of each page group. During the application execution, upon every period, the page scheduler performs model inference on the trained model of each page group and uses history-based

[1]Kleoniki is a Greek name, where 'niki' means victory.

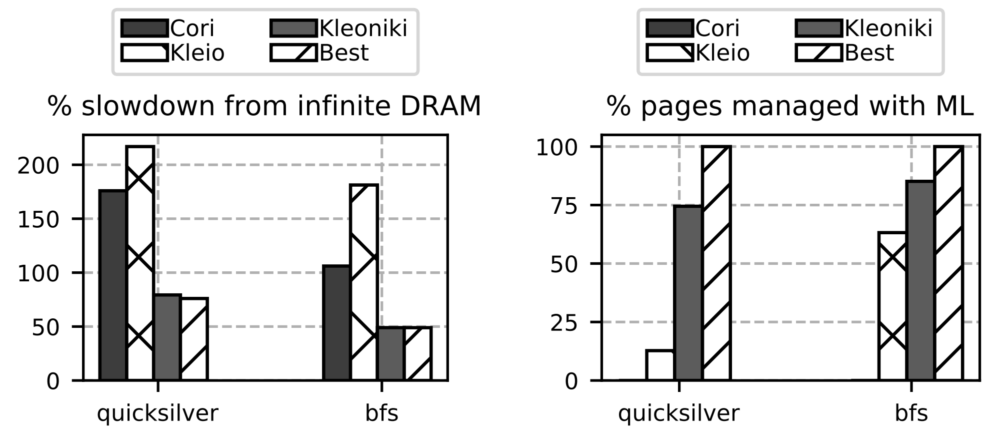

Figure 4.4: Preliminary results of performance and learning overheads associated with the proposed solution, Kleoniki. On the left figure, Kleoniki delivers best possible performance, due to the combination of machine intelligence and fine-tuned operational frequency. On the right, Kleoniki's effective page grouping enables drastically more pages to be managed intelligently under the *same* overheads with standalone Kleio.

information on the rest of the pages, as per Kleio's initial design.

4.3 Preliminary Results

Figure 4.4 shows preliminary results with two example workloads with very different data access behavior, that is regular for `quicksilver` and random for `bfs`. The operation of Kleoniki page scheduler at the frequency selected by Cori, together with the machine intelligent management of Kleio, realize best performance improvements for both application use cases. *This is not possible with standalone Cori due to the lack of machine intelligence, as well as for standalone Kleio due to its insight-less page scheduling frequency, where data movement overheads dominate.* Regarding the learning overheads, Kleoniki's effective page similarity grouping enables 75% of the application memory footprint to be managed intelligently, to deliver the aforementioned best application performance for `quicksilver`. For the same learning overheads, thus trained models, standalone Kleio can manage only 15% of the application pages. Yet, for `bfs` that has random accesses and dissimilar access patterns across pages, Kleoniki can only manage 10% more pages than Kleio, under the same overheads.

4.4 Summary

In conclusion, the absolute learning overheads of machine intelligent hybrid memory management highly depend on the application memory footprint and its regularity in access behavior. Appropriate page grouping can minimize such overheads via machine learning of access patterns that are similar across pages. This requires a mechanism to identify the similarity, as well as a revisit to the effectiveness of machine learning's accuracy across similar inputs. As a first step in this direction, the current approach groups pages with identical access patterns, whose inference from a single ML model with deliver the same accuracy levels. It leverages data reuse and period length insights from its previous contribution, Cori, to create large page groups with identical patterns of page access count across the scheduling periods. The proposed solution, Kleoniki, is a hollistic page scheduler that delivers maximum application performance due to the fine-tuned operational frequency and the machine intelligent management of drastically more pages than the prior contribution, Kleio.

CHAPTER 5
PROPOSED TIMELINE

5.1 Steps to Complete the Proposed Work

The remainder of this thesis will involve the following steps and exploration directions:

- Complete the preliminary results with evaluation of complete application suite, that establish the benefits of the page grouping mechanism across different data access behaviors.
- Extend the grouping mechanism to identify pages with similar access behavior. Explore the effectiveness of existing page grouping solutions based on locality, versus utilizing the proposed data reuse insights or even machine intelligent methods, such as k-means clustering.
- Explore the effects on machine learning model prediction accuracy across similar inputs. If the desired levels of accuracy are not met, further tune the neural network deployment or re-visit the page similarity approach.

5.2 Timeline

- **End of January**: Submit initial version of Kleoniki at HPDC 2021.
- **January - May**: Academic job search.
- **End of April**: Submit results from the exploration on page similarity and effects on machine learning accuracy at SC 2021.
- **April - May**: Thesis document composition.
- **June**: Thesis defense.

- **Summer ’21:** Graduation.

REFERENCES

[1] *Intel® Optane™ DC Persistent Memory*, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html, 2020.

[2] *MemVerge - More Memory. Less Cost.* https://www.memverge.com/more-memory-less-cost/, 2020.

[3] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, *Basic performance measurements of the intel optane dc persistent memory module*, 2019. arXiv: 1903.05714 `[cs.DC]`.

[4] D. Mhembere, D. Zheng, C. E. Priebe, J. T. Vogelstein, and R. Burns, "Knor: A numa-optimized in-memory, distributed and semi-external-memory k-means library," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '17, Washington, DC, USA: ACM, 2017, pp. 67–78, ISBN: 978-1-4503-4699-3.

[5] P. Wu, D. Li, Z. Chen, J. S. Vetter, and S. Mittal, "Algorithm-directed data placement in explicitly managed non-volatile memory," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '16, Kyoto, Japan: ACM, 2016, pp. 141–152, ISBN: 978-1-4503-4314-5.

[6] S. R. 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*, ser. EuroSys '16, London, United Kingdom: Association for Computing Machinery, 2016, ISBN: 9781450342407.

[7] D. Shen, X. Liu, and F. X. Lin, "Characterizing emerging heterogeneous memory," *SIGPLAN Not.*, vol. 51, no. 11, pp. 13–23, Jun. 2016.

[8] C. Cantalupo, V. Venkatesan, J. Hammond, K. Czurlyo, and S. D. Hammond, "Memkind: An extensible heap memory manager for heterogeneous memory platforms and mixed memory policies.," Mar. 2015.

[9] 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*, ser. SC '17, Denver, Colorado: ACM, 2017, 58:1–58:14, ISBN: 978-1-4503-5114-0.

[10] 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*, ser. SC '18, Dallas, Texas: IEEE Press, 2018, 31:1–31:13.

[11] Y. Chen, I. B. Peng, Z. Peng, X. Liu, and B. Ren, “Atmem: Adaptive data placement in graph applications on heterogeneous memories,” in *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, ser. CGO 2020, San Diego, CA, USA: Association for Computing Machinery, 2020, pp. 293–304, ISBN: 9781450370479.

[12] 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*, ser. ASPLOS ’17, China: Association for Computing Machinery, 2017, pp. 631–644, ISBN: 9781450344654.

[13] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, “Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, vol. 00, Feb. 2015, pp. 126–136.

[14] 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*, ser. ASPLOS ’19, Providence, RI, USA: Association for Computing Machinery, 2019, pp. 331–345, ISBN: 9781450362405.

[15] 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*, ser. ISCA ’17, Toronto, ON, Canada: Association for Computing Machinery, 2017, pp. 521–534, ISBN: 9781450348928.

[16] Z. Duan, H. Liu, X. Liao, H. Jin, W. Jiang, and Y. Zhang, “Hinuma: Numa-aware data placement and migration in hybrid memory systems,” in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 367–375.

[17] 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*, ser. ASPLOS ’16, Atlanta, Georgia, USA: Association for Computing Machinery, 2016, pp. 369–383, ISBN: 9781450340915.

[18] V. R. Kommareddy, S. D. Hammond, C. Hughes, A. Samih, and A. Awad, “Page migration support for disaggregated non-volatile memories,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS ’19, Washington, District of Columbia: Association for Computing Machinery, 2019, pp. 417–427, ISBN: 9781450372060.

[19] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, “Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 433–444.

[20] V. Gupta, M. Lee, and K. Schwan, “Heterovisor: Exploiting resource heterogeneity to enhance the elasticity of cloud platforms,” in *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE ’15, Istanbul, Turkey: Association for Computing Machinery, 2015, pp. 79–92, ISBN: 9781450334501.

[21] *MLPerf - Fair and useful benchmarks for measuring training and inference performance of ML hardware, software, and services.* https://mlperf.org/.

[22] *MLBench: Distributed Machine Learning Benchmark*, https://mlbench.github.io/.

[23] J. Li, Y. Ma, and R. Vuduc, *ParTI! : A parallel tensor infrastructure for multicore cpus and gpus*, Last updated: Jan 2020, Oct. 2018.

[24] M. Kulkarni, M. Burtscher, C. Casçaval, and K. Pingali, “Lonestar: A suite of parallel irregular programs,” in *ISPASS ’09: IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, USA, 2009.

[25] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, Jan. 2011.

[26] 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 *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC ’09, Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–54, ISBN: 978-1-4244-5156-2.

[27] *CORAL-2 Benchmarks*, https://asc.llnl.gov/coral-2-benchmarks/, 2020.

[28] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, “Coordinated and efficient huge page management with ingens,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’16, Savannah, GA, USA: USENIX Association, 2016, pp. 705–721, ISBN: 9781931971331.

[29] 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)*, 2017, pp. 152–165.

[30] *Pin - A Dynamic Binary Instrumentation Tool*, https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html, 2020.

[31] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, “Kleio: A hybrid memory page scheduler with machine intelligence,” in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC ’19, Phoenix, AZ, USA: ACM, 2019, pp. 37–48, ISBN: 978-1-4503-6670-0.

[32] T. D. Doudali, D. Zahka, and A. Gavrilovska, “Cori: Dancing to the right beat of periodic data movements over hybrid memory systems,” in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021.

[33] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, “Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using intel optane DC persistent memory modules,” in *Proceedings of the International Symposium on Memory Systems, MEMSYS 2019, Washington, DC, USA, September 30 - October 03, 2019*, ACM, 2019, pp. 288–303.

[34] S. Li, D. Reddy, and B. Jacob, “A performance amp; power comparison of modern high-speed dram architectures,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS ’18, Alexandria, Virginia, USA: Association for Computing Machinery, 2018, pp. 341–353, ISBN: 9781450364751.

[35] *Specifications - Supercomputer Fugaku : Fujitsu Global*, https://www.fujitsu.com/global/about/innovation/fugaku/specifications/, November 2020.

[36] *Summit - Oak Ridge Leadership Computing Facility*, https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/, 2020.

[37] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, “Disaggregated memory for expansion and sharing in blade servers,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA ’09, Austin, TX, USA: Association for Computing Machinery, 2009, pp. 267–278, ISBN: 9781605585260.

[38] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, “System-level implications of disaggregated memory,” in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, ser. HPCA ’12, USA: IEEE Computer Society, 2012, pp. 1–12, ISBN: 9781467308274.

[39] *Gen-Z Consortium: Computer Industry Alliance Revolutionizing Data Access*, https://genzconsortium.org/, 2020.

[40] *Intel Omni-Path Architecture (Intel OPA) Driving Exascale Computing and HPC Driving Exascale Computing and HPC with Intel*, https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html, 2020.

[41] G. Gill, R. Dathathri, L. Hoang, R. Peri, and K. Pingali, "Single machine graph analytics on massive datasets using intel optane dc persistent memory," *Proc. VLDB Endow.*, vol. 13, no. 8, pp. 1304–1318, Apr. 2020.

[42] I. B. Peng, M. B. Gokhale, and E. W. Green, "System evaluation of the intel optane byte-addressable nvm," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19, Washington, District of Columbia: Association for Computing Machinery, 2019, pp. 304–315, ISBN: 9781450372060.

[43] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18, Berlin, Germany: Association for Computing Machinery, 2018, pp. 41–42, ISBN: 9781450356299.

[44] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[45] *PolyBench/C - The Polyhedral Benchmark suite*, http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.

[46] D. Gureya, J. Neto, R. Karimi, J. Barreto, P. Bhatotia, V. Quema, R. Rodrigues, P. Romano, and V. Vlassov, "Bandwidth-aware page placement in numa," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 546–556.

[47] 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*, ser. MEMSYS '17, Alexandria, Virginia: Association for Computing Machinery, 2017, pp. 251–261, ISBN: 9781450353359.

[48] T. D. Doudali and A. Gavrilovska, "Mnemo: Boosting memory cost efficiency in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 412–421.

[49] L. Zhang, R. Karimi, I. Ahmad, and Y. Vigfusson, "Optimal data placement for heterogeneous cache, memory, and storage systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 1, May 2020.

[50] V. R. Kommareddy, A. Awad, C. Hughes, and S. D. Hammond, "Exploring allocation policies in disaggregated non-volatile memories," in *Proceedings of the*

Workshop on Memory Centric High Performance Computing, ser. MCHPC '18, Dallas, TX, USA: Association for Computing Machinery, 2018, pp. 58–66, ISBN: 9781450361132.

[51] C. Chou, A. Jaleel, and M. Qureshi, "Batman: Techniques for maximizing system bandwidth of memory systems with stacked-dram," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17, Alexandria, Virginia: Association for Computing Machinery, 2017, pp. 268–280, ISBN: 9781450353359.

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

[53] C. C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 1–12.

[54] M. Maas, D. G. Andersen, M. Isard, M. M. Javanmard, K. S. McKinley, and C. Raffel, "Learning-based memory allocation for c++ server workloads," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 541–556, ISBN: 9781450371025.

[55] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1919–1928.

[56] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '18, Boston, MA, USA: USENIX Association, 2018, pp. 759–773, ISBN: 978-1-931971-44-7.

[57] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18, Houston, TX, USA: Association for Computing Machinery, 2018, pp. 489–504, ISBN: 9781450347037.

[58] *autonuma: Optimize memory placement for memory tiering system*, https://lwn.net/Articles/835402/, October 27, 2020.

[59] *Instruction-Based Sampling: A New Performance Analysis Technique for AMD Family 10h Processors*, https://developer.amd.com/wordpress/media/2012/10/AMD_IBS_paper_EN.pdf, 16 November 2007.