Portable Application Guidance for Complex Memory Systems

M. Ben Olson
molson5@vols.utk.edu
University of Tennessee
Knoxville, Tennessee, USA

Brandon Kammerdiener
bkammerd@vols.utk.edu
University of Tennessee
Knoxville, Tennessee, USA

Michael R. Jantz
mrjantz@utk.edu
University of Tennessee
Knoxville, Tennessee, USA

Kshitij A. Doshi
kshitij.a.doshi@intel.com
Intel® Corporation
Chandler, Arizona, USA

Terry Jones
trjones@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

ABSTRACT

The recent emergence of new memory technologies and multi-tier memory architectures has disrupted the traditional view of memory as a single block of volatile storage with uniform performance. Several options for managing data on heterogeneous memory platforms now exist, but current approaches either rely on inflexible, and often inefficient, hardware-based caching, or they require expert knowledge and source code modification to utilize the different performance and capabilities within each memory tier. This paper introduces a new software-based framework to collect and apply memory management guidance for applications on heterogeneous memory platforms. The framework, together with new tools, combines a generic runtime API with *automated* program profiling and analysis to achieve data management that is both efficient and portable across multiple memory architectures.

To validate this combined software approach, we deployed it on two real-world heterogeneous memory platforms: 1) an Intel® Cascade Lake platform with conventional DDR4 alongside non-volatile Optane™ DC memory with large capacity, and 2) an Intel® Knights Landing platform high-bandwidth, but limited capacity, MCDRAM backed by DDR4 SDRAM. We tested our approach on these platforms using three scientific mini-applications (LULESH, AMG, and SNAP) as well as one scalable scientific application (QMCPACK) from the CORAL benchmark suite. The experiments show that portable application guidance has potential to improve performance significantly, especially on the Cascade Lake platform – which achieved peak speedups of 22x and 7.8x over the default unguided software- and hardware-based management policies. Additionally, this work evaluates the impact of various factors on the effectiveness of memory usage guidance, including: the amount of capacity that is available in the high performance tier, the input that is used during program profiling, and whether the profiling was conducted on the same architecture or transferred from a machine with a different architecture.

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357575

CCS CONCEPTS

• **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Software and its engineering** → *Runtime environments.*

KEYWORDS

memory management, heterogeneous memory, program profiling, application guidance, performance, Optane, 3DXPoint

ACM Reference Format:
M. Ben Olson, Brandon Kammerdiener, Michael R. Jantz, Kshitij A. Doshi, and Terry Jones. 2019. Portable Application Guidance for Complex Memory Systems. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3357526.3357575

1 INTRODUCTION

Memory technologies with very different performance and capabilities than conventional DDR SDRAM have begun to emerge. Some commercial platforms now package conventional memory DIMMs together with large capacity, non-volatile, memory devices, such as Intel®'s Optane™ DCPMMs[1] (3DXPoint) [15, 16], or alongside high bandwidth, but low capacity, "on-package" memory storage, such as MCDRAM [28]. These multi-tier memory architectures disrupt the traditional notion of memory as a single block of volatile storage with uniform performance. Figure 1 illustrates complex memory tiering in emerging memory architectures. Choosing the correct tier for data with high amounts of reuse can improve performance dramatically for full scale applications. As one works up the pyramid, access speed increases greatly, but capacity decreases, exacerbating the difficulty of intelligently backing allocations with a limited supply of fast memory.

These heterogeneous architectures require new data management strategies to utilize resources in different tiers efficiently. One common approach is to operate the faster, lower capacity tier(s) as a hardware-managed, memory-side cache. This approach allows applications to exercise high bandwidth memories at large data scale without requiring any source code modifications. However, hardware caching is inflexible, and can produce some unexpected and intractable inefficiencies for certain applications and access patterns. In this paper, we refer to this approach as *cache mode*.[2] An alternative approach, commonly referred to as software-based data tiering, allows the operating system (OS) to assign pages of

[1] DCPMM stands for Data Center Persistent Memory Module.
[2] It is also called *Memory Mode* in Intel®'s Optane™ DCPMM literature.

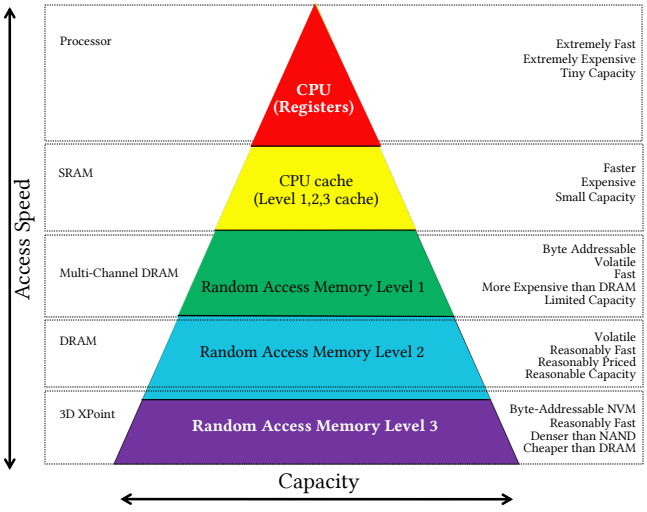


Figure 1: Data Tiers on Modern Complex Memory Platforms

memory to distinct memory tiers.[3] Without additional guidance from upper-level software, the OS may assign application data first to faster tier(s), and when faster memory is not available, to slower memory tiers. While this approach can also be applied to existing applications (with little or no software changes), it generally results in inconsistent and suboptimal performance – as pages accessed earlier or during periods of low demand are prematurely assigned into the high performance tiers even if they harbor lightly-accessed data. In this work, we refer to unguided software-based data tiering as the *first touch* approach.

The architecture and systems communities are currently exploring how to transcend limitations faced by common hardware- and software-based approaches. Several recent projects have proposed program profiling and analysis tools that augment applications with facilities to guide assignments of data to distinct memory tiers [7, 8, 10, 11, 26, 27, 30, 31]. For instance, the MemBrain approach [26] collects profiles of application memory usage and automatically converts them into tier recommendations for each data allocation site. While these studies demonstrate that application-level guidance is often much more effective than unguided approaches, their tools and frameworks are either only useful in simulation, or, like [26], have only been evaluated in limited usage scenarios on one type of heterogeneous architecture with a modest amount of upper tier memory.

This work aims to investigate the potential of generating and applying portable application guidance for programs that execute on different types of memory hardware. Towards this goal, we adopt and integrate the MemBrain approach into the Exascale Computing Project (ECP) Simplified Interface to Complex Memory (SICM) [4], which is a unified software framework for *automatically* adapting applications to a variety of memory devices and configurations. Using this combined framework, we evaluate the performance of data tier guidance on two platforms with very different heterogeneous memory architectures: 1) a "Knights Landing" (KNL[4]) platform with high-bandwidth MCDRAM backed by conventional DDR4, and 2) a "Cascade Lake" (CLX[5]) platform with DDR4 as the upper tier and high capacity Optane™ DC memory as the lower tier. To our knowledge, this is the first work to evaluate application guided memory management on a high volume commercial platform equipped with state-of-the-art DCPMM hardware.

Our experiments demonstrate the potential of portable application guidance using three mini-apps (LULESH, AMG, and SNAP) as well as one full scale application (QMCPACK) from the CORAL suite of high performance computing benchmarks [21]. Additionally, this work investigates the following high-level questions concerning profile-based guidance for heterogeneous memory management:

(1) How robust is it when different amounts of capacity are available in the high-performance tier?
(2) Is its performance sensitive to profile input?
(3) Can profiles collected on one architecture be used on a different architecture?
(4) How well does this portable, software-guided approach compete against unguided first touch and cache mode?

This work makes the following important contributions:

(1) It extends the SICM API and runtime to create an automated and unified application guidance framework for memory systems with different heterogeneous architectures.
(2) It shows that the SICM-based implementation achieves similar benefits as previous guidance-based approaches on the KNL, with speedups of up to 2.8x over unguided first touch.
(3) It finds that guided data tiering yields even larger performance gains on CLX, with maximum speedups of more than 22x and 7.8x over the unguided first touch and cache mode configurations, respectively.
(4) It shows that a single profile of a small program input is often just as effective for guiding execution with different and larger program inputs. Additionally, it finds that a single profile can guide execution effectively across different amounts of capacity available in the high performance memory tier.
(5) It demonstrates that profile guidance collected on one memory architecture may be transferred and applied favorably to a different architecture, but finds that there are limitations to this approach that can reduce its effectiveness.

2 BACKGROUND AND RELATED WORK

This section provides an overview of the SICM and MemBrain projects, which we combined to implement portable application guidance for complex memory systems. Later, it also discusses other recent research related to this project.

2.1 Simplified Interface to Complex Memory

The U.S. Department of Energy (DOE) is working towards achieving new levels of scientific discovery through ever-increasingly powerful supercomputers [5, 6]. Short-term plans call for achieving exaFLOP performance by the year 2021. To make these computing environments viable, the DOE has initiated a large effort titled the

[3] In Intel®'s ecosystem, this approach is called *Flat Mode* on Knights Landing, and *App-Direct* on Cascade Lake systems with Optane™ DCPMMs.

[4] Formally, Intel® Xeon® Phi™ 7250 Series Processors.
[5] Formally, Intel® Xeon® Gold 6262 Series Processors.

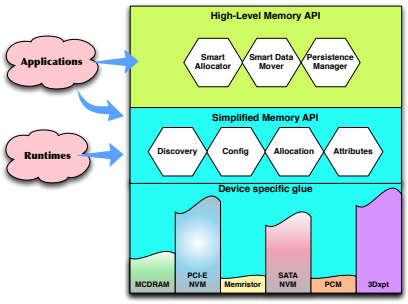


Figure 2: SICM overview [4]. The high-level provides a portable interface for applications, while the low-level implements efficient data management for complex memories.

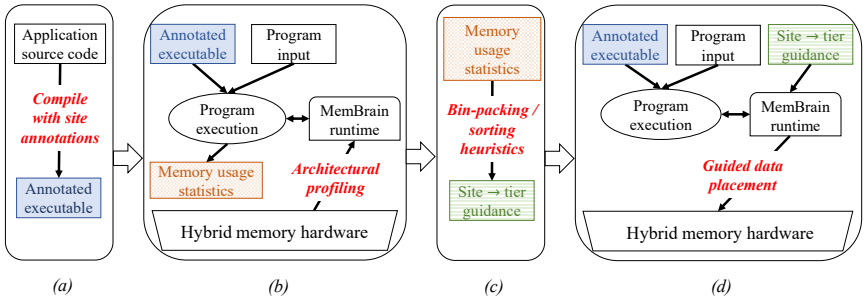


Figure 3: Application guidance workflow [26]. (a) Compile executable with source code annotations at each allocation site, (b) Profile memory usage of each site in a separate program run using architectural sampling, (c) Employ bin-packing / sorting heuristics to assign data-tier recommendations to each site, (d) Apply data-tiering recommendations during subsequent program executions.

Exascale Computing Project (ECP) [18, 24]. The project includes multiple thrust areas to deal with the hardware and software challenges of the most complex and high-performance supercomputers. For such systems, the DOE spends hundreds of millions of dollars to achieve the highest performance possible from available hardware.

The *Simplified Interface to Complex Memory* (SICM), one of the ECP projects, seeks to deliver a simple and unified interface to the emerging complex memory hierarchies on exascale nodes [4]. To achieve this goal, SICM is split into two separate interfaces: the low-level and the high-level, as shown in Figure 2. The high-level interface delivers an API that allows applications to allocate, migrate, and persist their data without detailed knowledge of the underlying memory hardware. To implement these operations efficiently, the high-level API invokes the low-level interface, which interacts directly with device-specific services in the OS. This work extends portions of both layers of SICM to enable automated and portable application guidance for complex memory architectures.

2.2 MemBrain: Automated Application Guidance for Hybrid Memory Systems

To integrate application guidance with SICM, this work embraces and extends the MemBrain approach [26]. MemBrain automates the use of data-tier guidance by associating profiles of memory behavior (such as bandwidth and capacity) with program *allocation sites*. Each allocation site corresponds to the source code file name and line number of an instruction that allocates program data (e.g., `malloc` or `new`) and may optionally include part or all of the call path leading up to the instruction. A separate analysis pass converts the profiles into tier recommendations for each site prior to guided execution. Figure 3 presents an overview of this approach.

2.2.1 Converting Site Profiles to Tier Recommendations. MemBrain includes three options for converting memory usage profiles into tier recommendations for each allocation site. The three options, which are also implemented in this work, are as follows:

Knapsack: The knapsack approach views the task of assigning application data into different device tiers as an instance of the classical 0/1 knapsack optimization problem. In this formulation, each allocation site is an item with a certain value (bandwidth) and weight (capacity). The goal is to fill a knapsack such that the total capacity of the items does not exceed some threshold (chosen as the size of the upper tier), while also maximizing the aggregate bandwidth of the selected items.

Hotset: The hotset approach aims to avoid a weakness of knapsack, namely, that it may exclude a site on the basis of its capacity alone, even when that site exhibits high bandwidth. Hotset simply sorts sites by their bandwidth per unit capacity, and selects sites until their aggregate size exceeds a soft capacity limit. For example, if the capacity of the upper tier is C, then hotset stops adding the sorted sites after the total weight is just past C. By comparison, knapsack will select allocation sites to maximize their aggregate value within a weight upper bound of C.

Thermos: Since hotset (intentionally) over-prescribes capacity in the upper tier, cold or lukewarm data could potentially end up crowding out hotter objects during execution. The thermos approach aims to address this occasional drawback. It only assigns a site to the upper tier if the bandwidth (value) the site contributes is greater than the aggregate value of the hottest site(s) it may displace. In this way, thermos avoids crowding out performance-critical data, while still allowing large-capacity, high-bandwidth sites to place a portion of their data in the upper-level memory.

2.3 Other Related Work

Propelled by computing trends such as Big Data and Exascale systems, research interest in application-guided data management has grown significantly in the last few years. Some recent works have combined program profiling and analysis with physical data management in operating systems and hardware to address a variety of issues on *homogeneous* memory platforms, including: DRAM energy efficiency [17, 25], cache pollution [14], NUMA traffic congestion [9], and data movement costs for non-uniform caches [23, 29]. While their techniques and goals are very different than this work, these studies highlight the power and versatility of application-level profiling for guiding data management.

Several other works have used program profiling to direct the assignment of data to different memory tiers in complex memory

architectures. For instance, some projects employ binary instrumentation to collect information about individual data objects and structures, and then use classification heuristics to assign data to the appropriate tier [7, 10, 11, 27, 30]. While this approach can be effective, instrumentation-based profiling is often slow due to the large number of data accesses generated by most applications, and may be inaccurate if hardware effects are not modeled properly. In contrast, this work employs architectural sampling to collect memory usage information with low overhead.

Some other projects have employed hardware support to collect data tiering guidance during execution [8, 20, 22]. However, these works only combine coarse-grained architecture-supported profiling with low-level (i.e., physical) memory management, and are therefore vulnerable to inefficiencies that arise from the applications working at cross-purposes from the OS and hardware. Unimem [31] also integrates hardware-based profiling with application events, and even adjusts tier assignments dynamically in response to shifting guidance. The adopted MemBrain approach applies static guidance, but unlike Unimem, it does not require any source code modification.

In contrast to all of these works, including MemBrain, this work is the first to study the use of common guidance infrastructure on multiple real complex memory platforms, including one with state-of-the-art non-volatile Optane™ DC memory. Using this framework, we also evaluate the robustness of profile guidance across several parameters, including the capacity of the upper tier, as well as the program input and architecture used during profiling.

3 PORTABLE APPLICATION GUIDANCE FOR COMPLEX MEMORY SYSTEMS

This section describes the extensions and changes this work makes to the SICM runtime to enable portable application guidance for complex memory systems.

3.1 Allocation Site Annotations

The original MemBrain toolset includes a static compilation pass, implemented in the LLVM toolchain [19]. It annotates allocation instructions with unique identifiers for each allocation site, so that sites may be identified and associated with memory usage profiles. To distinguish allocation instructions reached by multiple call paths, the pass furnishes an option to clone some number of layers of the function call path leading to each allocation instruction. For this work, we adopt and modify this LLVM-based tool to replace each allocation instruction with a corresponding call to the high-level SICM API with the site ID added in as a parameter.

New extensions also make it easier to modify existing applications as described next. The MemBrain annotation procedure requires multiple passes to: 1) convert each file to LLVM IR, 2) link all files to resolve function call context across the entire program, 3) perform site annotations, and 4) compile the annotated IR to executable code. For this work, we have abstracted these actions into wrapper commands for easy integration into build scripts for existing applications, as shown in an example `Makefile` in Listing 1. In most cases, including for all of the applications in this study, the build scripts require only a few lines of changes to generate an annotated executable file.

Listing 1: `Makefile` for building LULESH + site annotations. The changes for applying the annotations are shown in red.

```
export CXX_WRAPPER="$SICM_DIR/bin/cxx_wrapper.sh"
export LD_WRAPPER="$SICM_DIR/bin/ld_wrapper.sh"
...
all: $(LULESH_EXE)

.cc.o: lulesh.h
  $(CXX_WRAPPER) -c $(CXX_FLAGS) -o $@ $<

$(LULESH_EXE): $(OBJECTS)
  $(LD_WRAPPER) $(OBJECTS) $(LD_FLAGS) -o $@
```

3.2 Memory Usage Profiling

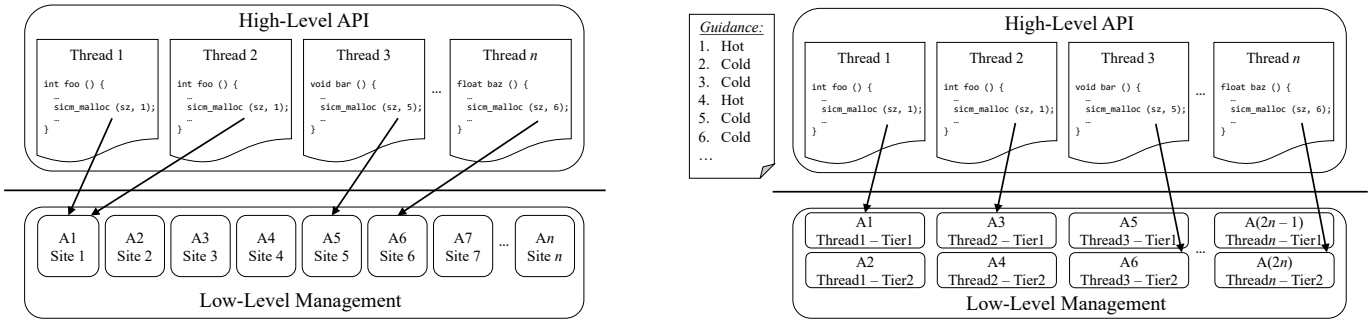
Similar to MemBrain, this work obtains profiling information during application execution through the use of low-overhead architecture-supported sampling. The profiles estimate the relative density of accesses of the data associated with each allocation site. While MemBrain requires a custom kernel module to collect samples of memory accesses, our framework implements a more portable technique that uses the Linux `perf` tool [3]. The tool allows user-level programs to customize the collection of hardware performance events and the subsequent analysis of the collection. For this study, we configured `perf` to collect samples of the relevant Precise Event-Based Sampling (PEBS) event that occurs when a data load misses the last level cache (LLC) on each platform. Specifically, we used the `MEM_LOAD_UOPS_RETIRED.LLC_MISS` event on the KNL platform, and the `MEM_LOAD_UOPS_RETIRED.L3_MISS` event on the CLX platform, and employ a sampling rate of $1/128$ events on both systems.

During profiling, the SICM runtime maintains shadow records of each allocation site. Each record includes a count of the number of `perf` samples corresponding to the addresses allocated by the site. In this way, the profiling tool constructs a *heatmap* of the relative access rates of each site. It also estimates the relative capacity of each site using a similar means as MemBrain; that is, it periodically counts the number of virtual pages associated with each site that are valid in physical memory using the Linux `pagemap` facility [2].

3.3 Arena Allocation

The low-level SICM runtime includes an arena allocator based on the standard `jemalloc` allocator [13]. By allocating program objects to page-aligned regions of virtual memory, arena allocation allows the runtime to manage and assign different groups of application data to distinct memory tiers independently of one another. We extended SICM's default allocator with two arena allocation schemes that enable the collection and use of application guidance.

Figure 4 illustrates the two arena allocation schemes. In these examples, the threads are running annotated code, in which the allocation instructions have been replaced with calls to the SICM API with the allocation site ID passed as their final argument. The first scheme, shown in Figure 4(a), creates a separate arena for each allocation site and shares those arenas between application threads. If two threads allocate data from the same site at the same time, one thread will block to avoid allocating multiple objects to the same address. While such (typically rare) contention can slow execution, this scheme simplifies profiling because it facilitates the assignment of sampled addresses to each allocation site.



(a) Thread-shared, per-site arenas for memory usage profiling.

(b) Thread-exclusive, per-tier arenas for guided execution.

Figure 4: Arena layouts in SICM. The layout in (a) facilitates the assignment of sampled addresses to each site profile, while the layout in (b) ensures each thread is able to allocate data to the correct tier without contention from the other threads.

For guided execution, the runtime creates a set of private arenas for each thread, where each set includes one arena for each memory tier, as shown in Figure 4(b). This approach allows the application to allocate data to different arenas corresponding to distinct tiers without requiring any synchronization across threads. For each allocation, the allocating thread first looks up the site ID in the (static) guidance file to determine the proper arena for the new data. To bind each arena to its corresponding device tier, SICM invokes the appropriate system interface whenever an arena is created or re-sized. On our Linux-based platforms, it uses the `mbind` system call with `MPOL_PREFERRED` to enforce arena-tier assignments.

3.4 Translating Profiles Across Platforms

An important advantage of our approach is that the platform that collects memory usage profiling does not necessarily need to match the platform on which it is deployed. Since the compiler pass associates each allocation site ID with file names and line numbers from high-level source code, many profiles can be transferred across different platforms without additional effort. However, in some cases, the context for an allocation site on the profiling platform might include language libraries or other supporting software that are not installed on the production platform. Configuration options and architectural differences can also change the way an application's source code is pre-processed and compiled on different platforms.

To address these issues, we developed a custom tool that attempts to match allocation contexts from executable files that were compiled in different software environments. For each allocation site in the executable file compiled on the source platform, the tool compares the strings describing the site's context with the allocation contexts on the destination platform. If no match is found, the site's profiling information is simply not mapped to the destination platform. In some cases, multiple sites from the source platform might match the same allocation context on the destination platform. For example, the compiler might create multiple copies of allocation sites within C++ template code. In these scenarios, the tool chooses the site that was accessed most frequently during the profiling run to ensure profiles of hot allocation sites are always transferred to the destination platform.

4 EXPERIMENTAL SETUP

4.1 Platform Details

Our evaluation employs two heterogeneous memory platforms. The first, which we refer to as the "Knights Landing" platform and abbreviate as KNL in this work, includes an Intel® Xeon® Phi™ 7250 processor with 68 quadruple hyper-threaded cores (272 hardware threads), clocked at 1.3GHz each, and a shared 34MB cache. Its memory system contains 16GB (8x2GB) of MCDRAM and 96GB (6x16GB) of 2400 MT/s DDR4 SDRAM. The upper MCDRAM tier sustains about 5x more bandwidth with similar access latencies as the lower DDR4 tier [28].

The second platform, called "Cascade Lake" and abbreviated as CLX, contains two Intel® Xeon® Gold 6262 processors, each with 24 compute cores (48 hardware threads) with a maximum clock speed of 3.6GHz and a shared 33MB cache. Each socket includes 192GB (6x32GB) of 2666 MT/s DDR4 SDRAM and 512GB (4x128GB) of non-volatile Optane™ DC memory hardware. For data reads, the Optane™ tier requires 2x to 3x longer latencies and sustains 30% to 40% of the bandwidth as the DDR4 memory. While latency for writes is similar on both tiers, the DDR4 tier supports 5x to 10x more write bandwidth than the Optane™ devices [16]. To avoid issues related to NUMA placement, all of our experiments use the processor and memory on only one socket of the CLX platform. We also installed recent Linux distributions as the base operating system for each platform: Debian 9.8 with kernel version 4.9.0-8 on the KNL, Fedora 27 with kernel version 5.1.0-rc4 on the CLX.

4.2 Workloads

Our evaluation employs three proxy applications (LULESH, AMG, and SNAP) as well as one full scale scientific computing application (QMCPACK) from the CORAL high performance computing benchmark suite [21]. The applications were selected based on their potential to stress cache and memory performance on our platforms. To study the impact of tiering guidance under different constraints and usage scenarios, we also constructed a set of inputs for each application on each platform. Table 1 presents descriptions of each selected application as well as usage statistics for each input on each platform with the unguided first touch configuration. Thus, the inputs we have selected generate a range of execution times

Table 1: Workload descriptions and statistics. Alongside the application name and description, the columns on the right show the arguments we used to construct each input as well as the execution time, figure of merit (throughput), peak resident set size (in GB), and # of allocation sites reached during execution of each input with the unguided first touch configuration.

Application	Description	Platform	Input	Input Arguments	Time	FoM	RSS (GB)	Sites
LULESH	Performs a hydrodynamics stencil calculation with very little communication between computational units. Represents the numerical algorithms, data motion, and programming style typical in scientific C/C++ applications. Version 2.0. **FoM metric: zones per second.**	KNL	Small	`-s 220 -i 12 -r 11 -b 0 -c 64 -p`	1.9m	1253.9	10.5	88
			Medium	`-s 340 -i 12 -r 11 -b 0 -c 64 -p`	7.2m	1,213.4	36.9	87
			Large	`-s 420 -i 12 -r 11 -b 0 -c 64 -p`	20.6m	769.8	69.7	87
		CLX	Small	`-s 220 -i 12 -r 11 -b 0 -c 64 -p`	1.8m	1,208.2	11.6	87
			Medium	`-s 520 -i 6 -r 11 -b 0 -c 64 -p`	14.5m	1,021.3	146.2	87
			Large	`-s 690 -i 3 -r 11 -b 0 -c 64 -p`	1.3h	220.2	339.7	87
			Huge	`-s 780 -i 3 -r 11 -b 0 -c 64 -p`	8.0h	50.0	492.9	87
AMG	Parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. Highly synchronous and generates a large amount of memory bandwidth. Version 1.0, written in ISO-C. **FoM metric:** $\frac{nnz*(iters+steps)}{seconds}$	KNL	Small	`-problem 2 -n 120 120 120`	0.6m	3.46E8	7.1	304
			Medium	`-problem 2 -n 220 220 220`	4.1m	3.13E8	43.9	478
			Large	`-problem 2 -n 270 270 270`	7.6m	3.11E8	80.0	397
		CLX	Small	`-problem 2 -n 120 120 120`	0.3m	5.37E8	3.9	215
			Medium	`-problem 2 -n 400 400 400`	13.4m	5.39E8	140.7	216
			Large	`-problem 2 -n 520 520 520`	1.1h	2.43E8	315.4	215
			Huge	`-problem 2 -n 600 600 600`	2.4h	1.67E8	477.2	218
SNAP	Mimics the computational requirements of PARTISN, a Boltzmann transport equation solver developed at LANL. Exhibits smooth scaling with a large number of program threads. Version 1.07, written in Fortran 90/95. **FoM metric: inverse of the grind time (ns).**	KNL	Small	nx=272, ny=32, nz=32	8.8m	1.28E−2	9.7	91
			Medium	nx=272, ny=136, nz=32	23.7m	2.03E−2	41.6	91
			Large	nx=272, ny=136, nz=68	34.9m	2.94E−2	88.4	91
		CLX	Small	nx=272, ny=32, nz=32	2.0m	5.81E−2	9.5	502
			Medium	nx=272, ny=136, nz=136	31.7m	6.39E−2	169.2	386
			Large	nx=272, ny=272, nz=120	4.1h	1.43E−2	301.7	276
			Huge	nx=272, ny=272, nz=192	10.5h	9.03E−3	485.2	91
QMCPACK	Quantum Monte Carlo simulations of the electronic structure of atoms and molecules. Exhibits near-perfect weak scaling and extremely low communication. Version 3.4, mostly written in C++. **FoM metric:** $\frac{blocks*steps*N_w}{seconds}$	KNL	Small	NiO S16 with VMC method, 272 walkers	3.1m	24.8	12.4	1750
			Medium	NiO S32 with VMC method, 272 walkers	16.6m	1.77	43.0	1809
			Large	NiO S32 with VMC method, 1088 walkers	54.4m	1.80	78.2	1852
		CLX	Small	NiO S32 with VMC method, 48 walkers	2.3m	2.27	10.6	1633
			Medium	NiO S128 with VMC method, 96 walkers	30.4m	5.69E−2	159.8	1425
			Large	NiO S128 with VMC method, 384 walkers	30.9h	3.45E−3	362.3	1609
			Huge	NiO S256 with VMC method, 48 walkers	45.6h	2.93E−4	475.4	1566

and capacity requirements. The smallest inputs complete in only a few minutes and fit entirely within the upper memory tier on each platform, while the largest inputs require multiple hours of execution time and would use almost all of the capacity available in the cache mode configuration.

4.3 Common Experimental Configuration

All applications were compiled using the LLVM compiler toolchain (v. 6.0.1) with default optimization settings and `-march=native`. C/C++ codes use the standard `clang` frontend, and Fortran codes are converted to LLVM IR using Flang [1]. All guided and non-guided configurations use SICM with the unmodified `jemalloc` allocator (v. 5.2.0) with `oversize_threshold` set to 0, and all other parameters set to default.[6] To prepare executables for guided execution, we configure the compilation pass to clone up to three layers of call path context to each allocation site. Recent work has shown that this amount of context is sufficient to obtain the benefits of this approach for most applications [12].

All workloads use OpenMP with one software thread for each hardware thread on each platform (i.e., 272 software threads on KNL, 48 on CLX), and one MPI rank, if applicable. For each experimental run, we execute the workload on an otherwise idle machine, and report the figure of merit (FoM) as the performance of the workload. Due to limited machine resources, and the large number of experiments in this study, we were only able to conduct one experimental run for most configurations of each workload.

[6]Setting `oversize_threshold` to 0 disables a feature of `jemalloc` that allocates objects larger than a specific size to a dedicated arena (to reduce fragmentation).

Table 2: Mean, minimum, and maximum FoM of first touch and cache mode over five runs. The minimum and maximum FoMs are shown relative to the mean.

Application	Input	First touch			Cache mode		
		Mean	Min	Max	Mean	Min	Max
LULESH	KNL: Small	1253.9	0.98	1.03	1305.9	0.97	1.02
	CLX: Med.	1021.3	0.99	1.00	911.0	0.98	1.01
AMG	KNL: Small	3.46e8	0.99	1.01	3.36e8	0.91	1.04
	CLX: Med.	5.39e8	0.99	1.00	5.30e8	0.99	1.00
SNAP	KNL: Small	0.0128	0.99	1.01	0.0125	0.99	1.01
	CLX: Med.	0.0639	0.98	1.02	0.0368	0.97	1.02
QMCPACK	KNL: Small	24.765	0.99	1.00	24.567	0.99	1.00
	CLX: Med.	0.0568	0.99	1.00	0.0484	0.99	1.01

To estimate the degree of variability in our results, we conducted five experimental runs for each of the small inputs on the KNL and the medium inputs on CLX with the first touch and cache mode configurations. Table 2 presents the mean FoM of these runs, as well as the minimum and maximum FoM relative to the mean, for each workload. Thus, variance for all of the workloads is relatively low on both architectures. We also find that cache mode exhibits slightly higher variability than first touch, perhaps due to timing and micro-architectural effects in the memory cache. In the limited number of cases we have evaluated, we have found that the variability of guided execution is similar to that of the first touch configuration.

All program profiling was conducted with hardware-managed caching enabled because we found this configuration runs significantly faster than the flat addressing mode for larger input sizes. Program data that originates from sites that are not reached during profiling is always assumed to be cold and assigned to the lower tier during guided execution. Non-heap (i.e., global and stack) data

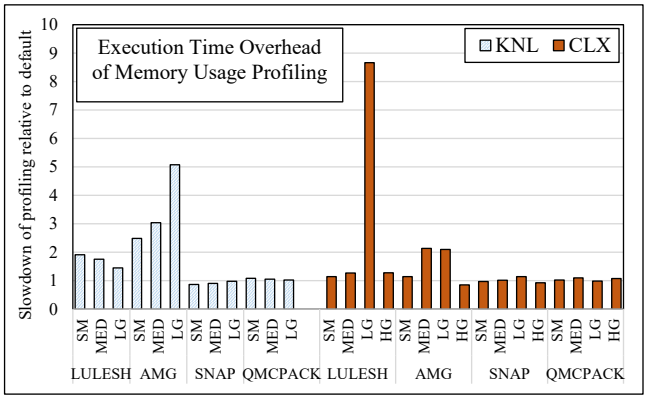


Figure 5: Performance (execution time) of each input on each platform in cache mode with profiling enabled relative to the default cache mode configuration (lower is better).

makes up a relatively small (< 2%) portion of the total memory footprint of all programs and inputs. All guided configurations assign non-heap data to the upper tier, if space is available.

Additionally, we found that the *hotset* approach for converting profiles to tier recommendations for guided execution performs as well or better than *knapsack* and *thermos* for the vast majority of our experiments. Thus, the results in the following section use the *hotset* approach unless otherwise noted.

5 EVALUATION

5.1 Overhead of Program Profiling

We first evaluate the overhead of collecting memory usage profiles in our framework. Figure 5 shows the execution time of each program input while profiling memory usage in cache mode relative to the default cache mode configuration. Thus, while some workloads exhibit higher overheads, average slowdowns are only 53% on KNL and 32% on CLX. We view these overheads as acceptable for the intended usage as an offline profile collection tool within a supercomputer center. In such environments, domain scientists often cycle between a development phase in which more life-like realism is introduced into their simulation applications, and a deployment phase in which results are collected from long-running jobs with a "production" release of the application. Since the long-running jobs may continue for many hours (in some cases, many days), the benefits of our workflow far exceed the cost of the overhead.

In the worst cases (LULESH LG, AMG LG), much of the performance loss is due to the use of the alternative arena allocation strategy (shown in Figure 4a) – which we confirmed by disabling profiling during a run with the alternative allocation strategy. We found that this configuration is 80% slower with the large input of AMG on KNL, and 8.2x slower for the large input of LULESH on CLX. Later in this section (figures 6–10), we show that profiles of smaller inputs are useful across different amounts of capacity in the upper tier and are typically transferable to larger executions. Therefore, despite the occasional higher overheads in Figure 5, we expect that this profiling technique can be adapted to an online and fully automatic guidance approach.

5.2 Robustness of Guidance with Different Amounts of Capacity in the Upper Tier

Our next experiments investigate the robustness of application guided data placement across different amounts of upper tier memory capacity. For this study, we employ a single input on each platform: *small* on KNL and *medium* on CLX. These inputs are small enough that the application's data always fits entirely within the upper tier. To generate and control contention for the high performance memory, we use a separate (otherwise idle) process to reserve some amount of upper tier capacity prior to launching the application. For guided execution, we employ profiles generated against the same program input for each configuration. Immediately prior to each guided run, the runtime applies one of the approaches described in Section 2.2.1 to compute site to tier recommendations for each given capacity assumed for the upper tier.

The line graphs in Figure 6 display the performance (FoM) of each application with both the guided and first touch approaches.[7] This study also omits cache mode because we could not exert direct control over the capacity of the in-memory caches. We plot performance for a range of upper tier capacities, each of which is presented as a percentage of the peak resident set size (RSS) of the application.[8] Each data point in Figure 6 and is relative to a baseline that places all application data in the lower tier.

We find that both the guided and unguided approaches perform better as upper tier capacity rises. At the same time, the guided approach makes use of the limited upper tier capacity much more efficiently than the unguided (first touch) allocation strategy. We also find that allocation guidance is often much more effective on the CLX platform than on the KNL platform. This result is due to the wider performance asymmetry between DDR4 and Optane™ DCPMMs, underscoring the greater salience of intelligent data tiering on the CLX platform.

Another interesting finding is that the guided approach typically achieves sharper and more pronounced performance gains at specific thresholds, while the unguided approach exhibits more gradual improvements as capacity increases. This effect arises because performance critical data typically originates from a single or small set of allocation sites. When there is enough capacity for the guided approach to assign these hot sites to the upper tier, performance improves markedly.

5.3 Effectiveness of Different Profile Inputs

Our next study evaluates the impact of different profile inputs on the effectiveness of guided data placement. For this study, we execute guided configurations of each workload with profiling collected from a run of the same program input as well as from runs of smaller inputs for the same application, without changing the size of the upper tier. We compare each guided configuration to both the unguided first touch and cache mode approaches.

KNL Platform: Figure 7 shows the performance of the cache mode and guided approaches for each program input relative to

[7] The *hotset* heuristic for dividing allocation sites into hot and cold sets exhibits suboptimal performance with AMG at 10% on CLX and with QMCPACK at 45% on KNL. These figures use the *thermos* approach for these cases, and use *hotset* for all others.
[8] We collected and show results for the CLX platform with larger capacity increments than the KNL due to time constraints on the CLX machine.

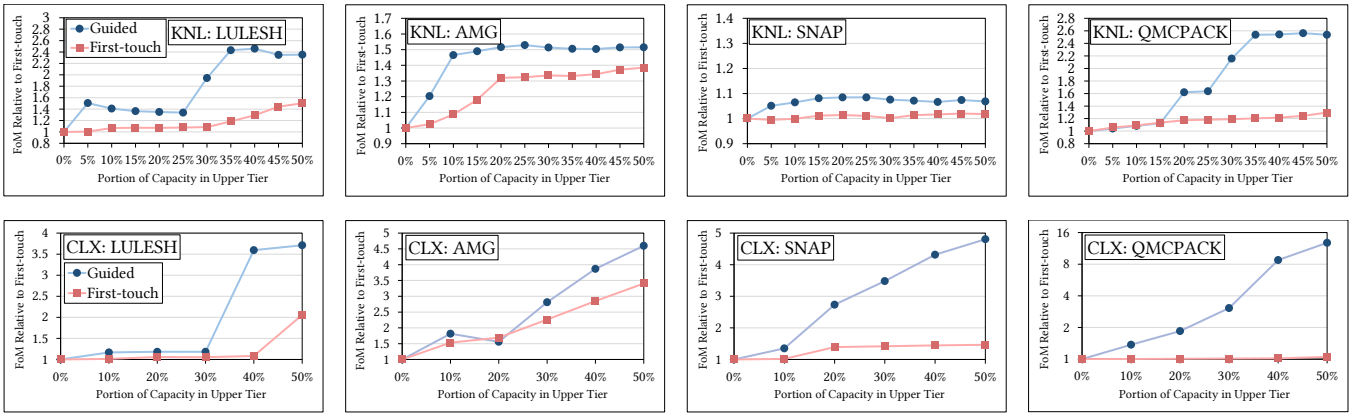


Figure 6: First touch and guided performance with different amounts of capacity in the upper tier on KNL and CLX. Each marker shows application performance with some amount of capacity available in the upper tier (expressed as a portion of the total capacity requirements) relative to a configuration that places all application data on the lower tier (higher is better).

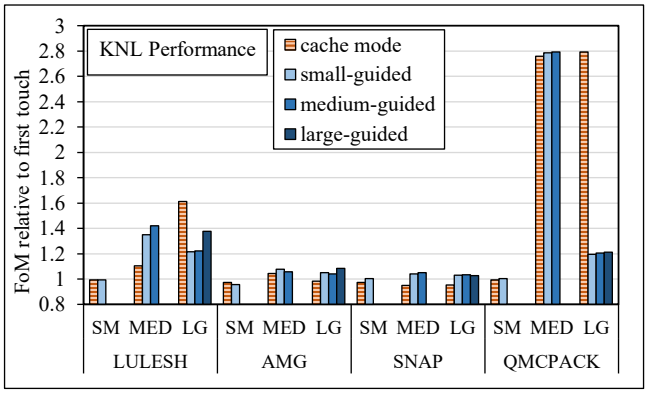


Figure 7: Performance (throughput) of each input with cache mode and guided approaches relative to first touch on KNL. The solid bars show results for guided execution with profiles of smaller and similar size inputs (higher is better).

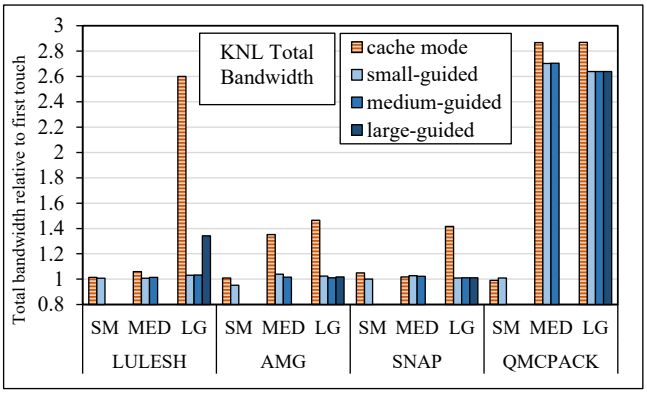


Figure 8: Total memory bandwidth of each program with each input with cache mode and guided approaches relative to first touch on the KNL platform.

first touch on the KNL platform.[9] The guided approach typically performs as well or better than first touch, but is exceeded in some cases by cache mode. The guidance-based approach computes tier recommendations offline, and does not attempt to migrate program data after its initial allocation. For applications that generate high bandwidth from different sets of program data during different execution periods, such static placement might be slower than an adaptive approach like cache mode. However, as shown in Figure 8, static placement typically generates less *total* bandwidth than cache mode and, as a result, is often more energy efficient.

We also find that profiles of smaller inputs are often just as effective as profiles of the same input for these applications. One notable exception is LULESH with the large program input, which runs about 17% faster with guidance from the large input profile compared to guidance from the medium input. In this case, we found that the medium profile accurately characterizes the relative access rates of each site, but but does not estimate the relative capacity requirements of each site accurately, resulting in less effective tier recommendations for the large input. Thus, one potential solution for generating better guidance for unseen program inputs is to combine offline profiles of access rates from known inputs with more accurate capacity estimates collected during guided execution.

CLX Platform: Figure 9 displays the performance of the cache mode and guided approaches with each input on the CLX platform.[10] Additionally, Figure 10 presents the average bandwidths of the first touch, cache mode, and guided approaches. On the CLX platform, the small inputs of each application exhibit similar performance for all guided and unguided configurations. To reduce clutter, we omit results in both figures for the small input size, and show only the guided approach with the medium input in Figure 10.

[9] All of the guided runs use the *hotset* approach except for the medium input of QMCPACK with medium profiling, which uses *thermos*.

[10] All guided runs use *hotset* except for the huge input of LULESH with large profiling and the huge input of QMCPACK with huge profiling, which both use *thermos*.

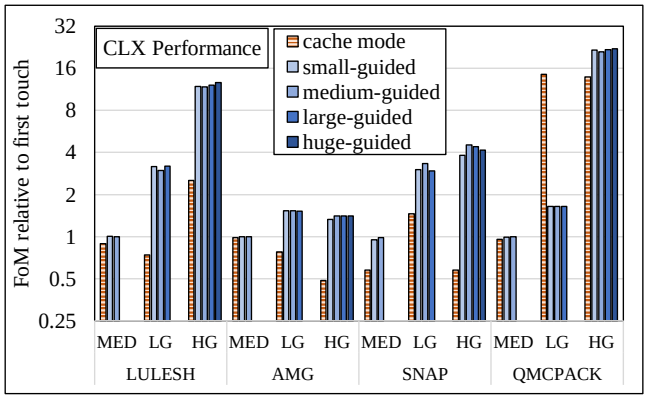


Figure 9: Performance (throughput) of each input with cache mode and guided approaches relative to first touch on CLX. The solid bars show results for guided execution with profiles of smaller and similar size inputs (higher is better).

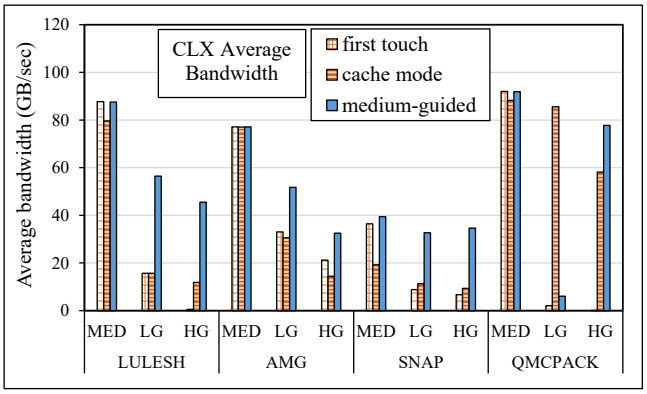


Figure 10: Average memory bandwidth of each program input with first touch, cache mode, and guided approach with medium input profile on the CLX platform.

As with our findings in Section 5.2, application guidance delivers a more significant impact on CLX (compared to KNL) due to the greater performance asymmetry between its memory tiers. For the unguided approaches, the medium inputs in Figure 9 generate relatively high average bandwidths because all of their data fits within the upper tier. Average bandwidth for the larger inputs is typically lower because more data is allocated to the Optane™ DC tier. The guided approaches are comparatively more resilient because they push a larger portion of the performance critical data into the DDR tier.

At the same time, profiles based on the small program inputs enable most, or all, of the benefits of guided data placement for larger input sizes. In the best case, we found that profile of the small input of QMCPACK, which executes for less than three minutes, speeds up the huge input by more than 21.5*x* compared to unguided first touch. Additionally, static guidance frequently outperforms the hardware-based cache mode, achieving, for example, a peak speedup of over 7.8*x* with the huge input of SNAP. Only one instance, QMCPACK with the large input size, exhibits worse performance with the guidance approaches than with cache mode. On further analysis, we found this result is related to the method by which we constructed this input, as described next.

QMCPACK includes several test input files that generate a range of processing and capacity requirements. For instance, the medium input employs the `NiO S128` test file, which simulates the electronic structure of 128 atoms with 1,536 electrons. Since QMCPACK did not include a test input file with computational requirements similar to those of the other large inputs, we constructed the large input by increasing the number of processing units (known as walkers) for the `NiO S128` test case. While this approach does increase the time and space demands of the application, it also changes the proportion of data created at each allocation site. For example, we found that one of the workload's relatively "hot" allocation sites corresponds to only 12% of the total capacity when we use a smaller number of walkers for the medium input, but the same site generates over 70% of the capacity when we increase the number of walkers for the large input. In this case, the guided approach is not able to assign such a large proportion of hot data from this site to the upper tier, and therefore lags behind the performance of cache mode.

5.4 Performance of Using Different Platforms for Profiling and Guided Execution

Lastly, we examine the potential of using profiles collected on one platform to generate guidance for systems with different memory architectures. Such a capability would allow users to profile application behavior on earlier or less resource-rich platforms, and use it to guide data placement on systems with memory that is more expensive, in higher demand, or earmarked for production software. For these experiments, we employ the profile conversion tool described in Section 3.4 to translate profiles of the small program input from the KNL to the CLX platform, and vice versa.

Figures 11 and 12 present the performance of the guided approach with each input on each platform between guidance obtained by two types of profiling: 1) native profiling from the same platform, and 2) cross-platform profiling obtained by conversion from the other platform. In many cases, the cross-platform profiling obtains all, or a significant portion, of the benefits of native profiling. For the others, profiles from a different architecture fall short of delivering the impact of profiling on the same platform. For instance, the huge inputs of LULESH and QMCPACK on CLX (Figure 12) exhibit some benefits with the cross-profiling drawn from the KNL, but are still about 2*x* to 5*x* slower than guided execution with profiling from the same system.

There are multiple factors that could limit the effectiveness of the guided approach with cross-platform profiling. Specifically, 1) The program input or number of threads that are used during profiling on the source platform deviate significantly from those used during execution on the target platform, 2) Hardware-based profiling on older architectures, such as KNL, can be less accurate than architectural profiling on the target platform, and 3) Cross-platform profile conversion is lossy; it is not always possible to find exact correspondence between the allocation contexts due to differences between system libraries and supporting software. This last factor is particularly important on the platforms we used for this study, which require different Linux distributions and include different

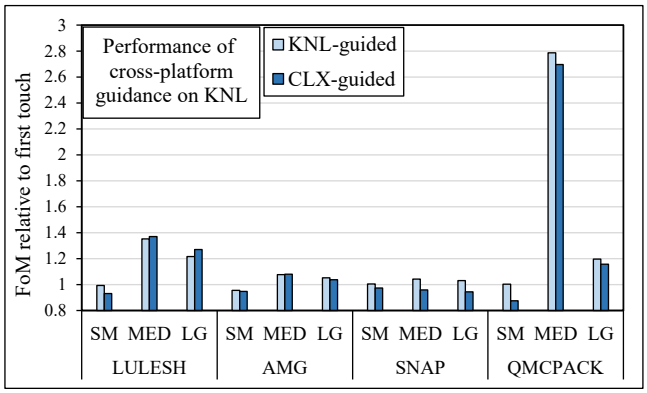


Figure 11: Performance comparison of guided execution on the KNL platform with profiling collected on the same KNL platform and on CLX (higher is better).

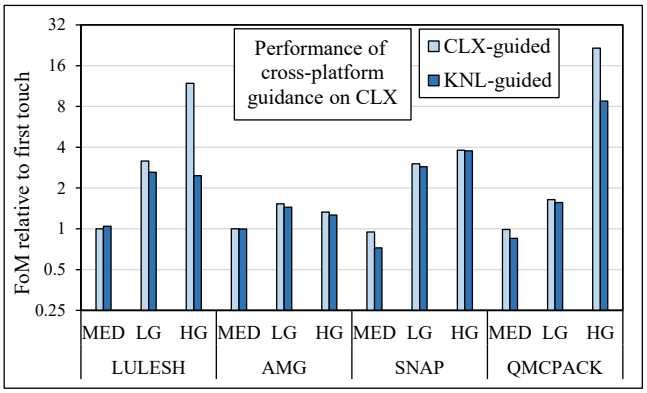


Figure 12: Performance comparison of guided execution on the CLX platform with profiling collected on the same CLX platform and on KNL (higher is better).

versions of the standard libraries for C, C++, and Fortran. Such unmatched contexts are conservatively assigned to the lower tier in our conversion process. In the future, we plan to setup identical software environments across platforms and conduct additional experiments to isolate, evaluate, and address the impact each of these factors individually.

6 FUTURE WORK

There are several paths for future work. Our studies show that application guidance from a separate profile run is often effective across a wide range of upper tier capacities and even with usage profiles collected during execution with different, and much smaller, program inputs. In the future, we will develop and integrate into SICM static program analyses and lightweight online profiling to generate and apply memory usage guidance without prior offline profiling. At the same time, we will study the performance of application guidance with a wider range of applications and program inputs to better understand the limitations of our approach. We have also found that some applications perform better with hardware-managed caching than with software-directed data placement, especially on systems with a limited amount of high bandwidth memory in the upper tier. To facilitate the use of software guidance with existing hardware features, we plan to design and implement new data characterization tools that automatically identify objects and usage patterns that work well with hardware caching. Finally, this study targets two Intel®-based platforms, one with high bandwidth on-package DRAM, and another with large capacity, non-volatile memory. As we take this work forward, we will modify our framework for use with other architectures and emerging technologies, including systems with three or more tiers of distinct memory hardware, and explore the potential challenges and opportunities that arise from guiding data management on more complex memory platforms.

7 CONCLUSIONS

This paper presents a unified software framework that combines a generic runtime API for emerging memory technologies with automated program profiling and analysis to enable efficient and portable application guided data management for complex memory hardware. It deploys and validates this approach on two Intel®-based server machines with real heterogeneous memory technology: one with conventional DDR4 alongside non-volatile memory with large capactiy, and another with limited capacity, high bandwidth DRAM backed by DDR4 SDRAM. The evaluation shows that application guidance can enable substantial performance and efficiency gains over default unguided software- and hardware-based schemes, especially on systems with state-of-the-art non-volatile memory hardware. Additional experiments reveal that a single profile of a smaller program input is often sufficient to achieve these gains across different amounts of capacity in the upper tier and with larger program inputs, in most cases. Furthermore, this work demonstrates that memory usage profiles collected on a machine with one type of memory hardware can be automatically and favorably adapted to platforms with a different architecture. Overall, the results show that there is great potential for this portable application guidance approach to address the challenges posed by emerging complex memory technologies and deliver their benefits to a wide range of systems and applications.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their thoughtful comments and feedback. We also thank Intel® DCG for providing the computing platform infrastructure that we used to conduct this study. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, as well as contributions from the National Science Foundation (under CCF-1617954) and the Software and Services Group (SSG) at Intel® Corporation.

REFERENCES

[1] [n.d.]. Flang. https://github.com/flang-compiler/flang
[2] [n.d.]. pagemap, from the userspace perspective. https://www.kernel.org/doc/Documentation/vm/pagemap.txt
[3] 2019. https://perf.wiki.kernel.org/index.php/Main_Page
[4] 2019. SICM: Simplified Interface to Complex Memory - Exascale Computing Project 2019. https://www.exascaleproject.org/project/sicm-simplified-interface-complex-memory/

[5] 2019. U.S. Department of Energy and Cray to Deliver Record-Setting Frontier Supercomputer at ORNL. https://www.ornl.gov/news/us-department-energy-and-cray-deliver-record-setting-frontier-supercomputer-ornl

[6] 2019. U.S. Department of Energy and Intel to deliver first exascale supercomputer. https://www.anl.gov/article/us-department-of-energy-and-intel-to-deliver-first-exascale-supercomputer

[7] Neha Agarwal, David Nellans, Mark Stephenson, Mike O'Connor, and Stephen W. Keckler. 2015. Page Placement Strategies for GPUs Within Heterogeneous Memory Systems. *SIGPLAN Not.* 50, 4 (March 2015), 607–618. https://doi.org/10.1145/2775054.2694381

[8] Neha Agarwal and Thomas F. Wenisch. 2017. 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 '17)*. ACM, New York, NY, USA, 631–644. https://doi.org/10.1145/3037697.3037706

[9] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. 2013. Traffic management: a holistic approach to memory placement on NUMA systems. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 381–394.

[10] Subramanya R Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 15.

[11] T. Chad Effler, Adam P. Howard, Tong Zhou, Michael R. Jantz, Kshitij A. Doshi, and Prasad A. Kulkarni. 2018. On Automated Feedback-Driven Data Placement in Hybrid Memories.. In *LNCS International Conference on Architecture of Computing Systems (ARCS '18)*.

[12] T. Chad Effler, Brandon Kammerdiener, Michael R. Jantz, Saikat Sengupta, Prasad A. Kulkarni, Kshitij A. Doshi, and Terry Jones. 2019. Evaluating the Effectiveness of Program Data Features for Guiding Data Management. In *Proceedings of the International Symposium on Memory Systems (MemSys '19)*. ACM, New York, NY, USA.

[13] Jason Evans. 2006. A Scalable Concurrent malloc (3) Implementation for FreeBSD. (2006).

[14] Rentong Guo, Xiaofei Liao, Hai Jin, Jianhui Yue, and Guang Tan. 2015. NightWatch: integrating lightweight and transparent cache pollution control into dynamic memory allocation systems. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 307–318.

[15] Intel. 2019. Intel Optane DC Persistent Memory. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[16] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR* abs/1903.05714 (2019). arXiv:1903.05714 http://arxiv.org/abs/1903.05714

[17] Michael R. Jantz, Forrest J. Robinson, Prasad A. Kulkarni, and Kshitij A. Doshi. 2015. Cross-layer Memory Management for Managed Language Applications. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015)*. ACM, New York, NY, USA, 488–504. https://doi.org/10.1145/2814270.2814322

[18] D. Kothe, S. Lee, and I. Qualters. 2019. Exascale Computing in the United States. *Computing in Science Engineering* 21, 1 (Jan 2019), 17–29. https://doi.org/10.1109/MCSE.2018.2875366

[19] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–. http://dl.acm.org/citation.cfm?id=977395.977673

[20] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu. 2017. Utility-Based Hybrid Memory Management. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*.

[21] LLNL. 2014. CORAL Benchmark Codes. https://asc.llnl.gov/CORAL-benchmarks.

[22] M.R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G.H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. 126–136. https://doi.org/10.1109/HPCA.2015.7056027

[23] Anurag Mukkara, Nathan Beckmann, and Daniel Sanchez. 2016. Whirlpool: Improving Dynamic Cache Management with Static Data Classification. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 113–127. https://doi.org/10.1145/2872362.2872363

[24] Barack Obama. 2015. Executive Order – Creating a National Strategic Computing Initiative. https://obamawhitehouse.archives.gov/the-press-office/2015/07/29/executive-order-creating-national-strategic-computing-initiative

[25] Matthew Benjamin Olson, Joseph T. Teague, Divyani Rao, Michael R. JANTZ, Kshitij A. Doshi, and Prasad A. Kulkarni. 2018. Cross-Layer Memory Management to Improve DRAM Energy Efficiency. *ACM Trans. Archit. Code Optim.* 15, 2, Article 20 (May 2018), 27 pages. https://doi.org/10.1145/3196886

[26] M Ben Olson, Tong Zhou, Michael R Jantz, Kshitij A Doshi, M Graham Lopez, and Oscar Hernandez. 2018. MemBrain: Automated Application Guidance for Hybrid Memory Systems. In *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE, 1–10.

[27] H. Servat, A. J. PeÃśa, G. Llort, E. Mercadal, H. Hoppe, and J. Labarta. 2017. Automating the Application Data Placement in Hybrid Memory Systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*.

[28] Avinash Sodani. 2015. Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor. In *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 1–24.

[29] Po-An Tsai, Nathan Beckmann, and Daniel Sanchez. 2017. Jenga: Software-Defined Cache Hierarchies. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 652–665. https://doi.org/10.1145/3079856.3080214

[30] GR Voskuilen, MP Frank, SD Hammond, and AF Rodrigues. [n.d.]. Evaluating the Opportunities for Multi-Level Memory–An ASC 2016 L2 Milestone. ([n. d.]).

[31] Kai Wu, Yingchao Huang, and Dong Li. 2017. 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 '17)*. ACM, New York, NY, USA, Article 58, 14 pages.