Demystifying the Performance of HPC Scientific Applications on NVM-based Memory Systems

Ivy Peng
Lawrence Livermore National Laboratory
Livermore, USA
peng8@llnl.gov

Kai Wu
University of California, Merced
Merced, USA
kwu42@ucmerced.edu

Jie Ren
University of California, Merced
Merced, USA
jren6@ucmerced.edu

Dong Li
University of California, Merced
Merced, USA
dli35@ucmerced.edu

Maya Gokhale
Lawrence Livermore National Laboratory
Livermore, USA
gokhale2@llnl.gov

***Abstract*—The emergence of high-density byte-addressable non-volatile memory (NVM) is promising to accelerate data- and compute-intensive applications. Current NVM technologies have lower performance than DRAM and, thus, are often paired with DRAM in a heterogeneous main memory. Recently, byte-addressable NVM hardware becomes available. This work provides a timely evaluation of representative HPC applications from the "Seven Dwarfs" on NVM-based main memory. Our results quantify the effectiveness of DRAM-cached-NVM for accelerating HPC applications and enabling large problems beyond the DRAM capacity. On uncached-NVM, HPC applications exhibit three tiers of performance sensitivity, i.e., insensitive, scaled, and bottlenecked. We identify *write throttling* and *concurrency control* as the priorities in optimizing applications. We highlight that concurrency change may have a diverging effect on read and write accesses in applications. Based on these findings, we explore two optimization approaches. First, we provide a prediction model that uses datasets from a small set of configurations to estimate performance at various concurrency and data sizes to avoid exhaustive search in the configuration space. Second, we demonstrate that write-aware data placement on uncached-NVM could achieve 2x performance improvement with a 60% reduction in DRAM usage.**

***Keywords*-Non-volatile memory; Optane; heterogeneous memory; persistent memory; byte-addressable NVM; HPC;**

I. Introduction

Byte-addressable non-volatile memories, such as STT-RAM, ReRAM, and PCM [11, 15, 24, 25], are promising to accelerate data- and compute-intensive HPC applications [16]. High-density NVM enables larger memory capacity than DRAM under the same area constraints. Data stored in NVM can persist through power failures as if in storage. Recently, some NVM technologies may even provide comparable bandwidth and latency to that of DRAM, enabling much higher performance than block devices. Altogether, these characteristics start blurring the boundary between memory and storage when NVM is used in the main memory. However, NVM technologies are still under active development and not ready for replacing DRAM. For instance, the write bandwidth of the Intel Optane DC persistent memory is only one third that of DRAM [21]. Consequently, NVM is often paired with DRAM, building a heterogeneous memory system.

The recent release of the Intel Optane DC persistent memory module (named `Optane` in the rest of the paper) marks the first mass production of byte-addressable NVM. The `Optane` provides a realistic and accessible hardware platform for evaluating the impact of new main memory designs on HPC scientific applications. The future Exascale system is reported to be based on an NVM technology like `Optane` [14]. Therefore, the performance of HPC applications on this new hardware requires a timely and comprehensive evaluation. Several works have provided system evaluation and performance of specific applications [9, 12, 21, 32]. Still, the landscape of HPC scientific applications requires a systematic approach to identify bottlenecks and opportunities. Does an NVM-based main memory change the priority in optimization? How to effectively leverage the heterogeneity in DRAM/NVM systems for the best performance? Answering these questions not only helps to exploit NVM on the next generation supercomputers but also influences the design of runtime and system software to accommodate this emerging memory technology.

In this work, we follow the well-known Seven Dwarfs [1] and choose flagship libraries and applications, such as ScaLAPACK [4], SuperLU [17], and Hypre [34] to cover the landscape of scientific applications. Our study provides a comprehensive evaluation of the domains of dense and sparse linear algebra, spectral methods, N-body methods, structured and unstructured grids, and Monte Carlo-based algorithms. We find that scientific applications exhibit three tiers of sensitivity on the uncached-NVM, i.e., insensitive, scaled, and bottlenecked. Leveraging DRAM as a cache to NVM could effectively improve application performance even when the input problems have a memory footprint three to five times the DRAM capacity. Furthermore, we reveal two bottlenecks arising from the asymmetric bandwidth and scaling limitation in NVM, i.e., *write-throttling* and *concurrency contention*. These characteristics may change the critical computation phases in applications and, thus, require different priorities in optimization. Also, we identify that concurrency changes have a *diverging effect* on read and write accesses in applications, which requires different strategies like the write-aware placement. We believe that this work provides insights and feedbacks that are critical for applications to leverage future systems with NVM-based main memory.

We explore two optimization directions. We develop a model to predict application performance in cached-NVM at different concurrency and problem sizes to help design space exploration and identify optimal configurations. On uncached-NVM, we demonstrate in ScaLAPACK that explicitly managing write-aware placement can significantly improve performance and reduce DRAM usage. We summarize our contributions as follows.

- A comprehensive performance study of HPC workloads from common computation domains (the Seven Dwarfs) on cached and uncached NVM-based main memory;
- Highlight that write throttling and concurrency contention change the priority of optimizing computation in scientific applications;
- Identify the diverging effect of concurrency change on read and write in applications, and demonstrate the effectiveness of write-aware data placement.
- Develop a prediction model to estimate performance at various concurrency and data sizes to select optimal configurations.

II. Background

In this section, we introduce NVM-based memory systems and the Seven Dwarfs in scientific applications.

A. NVM-based Heterogeneous Memory

Extensive research has proposed using NVM for implementing the main memory to exploit its high density, persistence, and power efficiency [15, 24]. Still, the current NVM technologies have lower performance than DRAM, and thus, main memory designs often pair NVM with DRAM, either as a cache or placed side-by-side to NVM. Previous works mostly use simulations and small problems for evaluation due to the lack of large-scale hardware. Recently, the first mass production of byte-addressable NVM arrives in the format of the Intel Optane DC Persistent Memory Module (PMM). In this work, we use this new hardware to evaluate realistic problems on promising memory designs.

The work of [21] has provided detailed system evaluation, and we briefly summarize the system architecture (Figure 1) in this section. The memory subsystem consists of DRAM DIMMs and NVDIMMs that share integrated memory controllers (iMC). Each NVDIMM has a small internal controller for address translation and a data buffer. The internal data granularity in the Optane media is 256 bytes, while the data granularity between the processor and memory subsystem is 64 bytes. System evaluation has quantified that sequential and random read accesses to NVM have a latency of 174 ns and 304 ns, respectively [21]. Write latency to NVM depends on store instructions and data sizes. For instance, 64- to 256-byte non-temporal data store has 180 – 200 ns latency [12]. On one socket, the read bandwidth to NVM can reach 39 GB/s while the peak write bandwidth is only 13 GB/s [12, 21]. Thus, the NVM exhibits about three times asymmetry in read and write bandwidth.

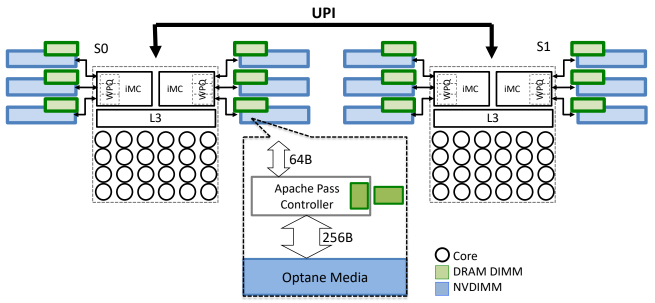


Figure 1: The system architecture the Intel Purley platform.

The NVDIMMs can be configured in *Memory* or *AppDirect* mode. In Memory mode, DRAM becomes a hardware-managed direct-mapped write-back cache to NVM and is transparent to applications. Note that DRAM on one socket cannot cache accesses to NVM on another socket [9]. In AppDirect mode, the NVM becomes a byte-addressable persistent memory. A *dax*-aware file system would transparently convert file read and write operations into 64-byte load and store instructions in this mode to access NVM. Also, in this mode, the NVM on each socket can be exposed as a non-uniform memory access (NUMA) node to the CPUs. Standard NUMA management routines like *numactl* can be used to control data placement in this configuration.

B. Seven Dwarfs of HPC scientific applications

The work of [1] summarizes seven domains of numerical algorithms in major HPC science and engineering applications, known as "Seven Dwarfs". For a comprehensive evaluation of the HPC landscape, we select one application from each Dwarf as well as Laghos [7], a proxy application of the BLAST hydrodynamics application, for the experiments. We introduce each Dwarf and application as follows.

- **Dense Linear Algebra** features dense array data structures. They exhibit strided memory access to all the elements of the data structures. Classic vector and matrix operations fall into this category. We select matrix

multiplication (level 3) from ScaLAPACK [4] for the experiment.
- **Sparse Linear Algebra** methods store data in compressed formats and access data elements through indirect memory accesses. We choose SuperLU [17] that adopts the BAR method for implementing sparse LU factorization.
- **Spectral Methods** often use fast Fourier transforms (FFT) to solve differential equations. Data permutation in this method often requires matrix transpose. We evaluate the FT benchmark that performs discrete 3D FFT from the NPB [2] suite.
- **N-Body Methods** have a high computation complexity of $\mathcal{O}(N^2)$ for simulating a dynamical system of N particles. We use hardware accelerated cosmology code (HACC) [10] that simulates the formation of structure in collisionless fluids under the influence of gravity in an expanding universe.
- **Structured Grids** feature regular grid structures. Stencil operations on the grids often have high spatial locality in data accesses. We choose Hypre [34], a high-performance pre-conditioners library for solving linear systems in our evaluation.
- **Unstructured Grids** feature irregular grid structures. Data accesses and updates often involve multiple levels of memory reference indirection. We use a general block-structured AMR framework, BoxLib [3], for the test.
- **Monte Carlo** methods rely on repeated random data accesses to calculate numerical results. We use XS-Bench [27], which implements a Monte Carlo neutron transport algorithm, as a representative of such workloads.

III. Methodology

In this section, we describe the experimental setup, benchmarks, and methodologies. We use the Intel Purley platform that consists of two 2nd Gen Intel® Xeon® Scalable processors as the testbed. The memory subsystem consists of four iMCs, 12 memory channels, a total of 192 GB DRAM (12 DIMMs), and 1.5 TB NVM (12 NVDIMMs). The memory channels run at 2400 GT/s, supporting 230.4 GB/s peak system bandwidth. We override the EFI memory map to expose NVM on each socket as a separate NUMA node. The configuration of the system is summarized in Table I.

The platform runs the Fedora 29 operating system with GNU/Linux 5.1.0. When the Optane DC PMM is configured in AppDirect mode and exposed as NUMA nodes, we use *numactl* to control the data placement onto different memories. Table II summarizes the applications and their input problems. We compile all applications with GCC 8.3.1. For each application, we report the application-defined figure of metric (FoM) if available. Otherwise, we report the run time of the main computation kernels.

We develop profiling routines that sample memory bandwidth on each NVDIMM and DRAM DIMM. We use the Intel Processor Counter Monitor (PCM) tool [26] to monitor hardware counters to collect core activities and offcore events. The profiling routines are integrated into applications to exclude the initialization and finalization stages. We only report the profiling results of the main computation phases. When the Optane is configured in Memory mode, we report the memory traffic as measured to DRAM DIMMs because it is accessed before NVDIMMs. When the Optane is in AppDirect mode, we report the memory traffic as the sum of traffic to NVDIMMs and DRAM DIMMs.

Table I: Platform Specifications

Processor	2nd Gen Intel® Xeon® Scalable processor
Cores	2.4 GHz (3.9 GHz Turbo frequency × 24 cores (48 HT) × 2 sockets
L1-icache	private, 32 KB, 8-way set associative, write-back
L1-dcache	private, 32 KB, 8-way set associative, write-back
L2-cache	private, 1MB, 16-way set associative, write-back
L3-Cache	shared, 35.75 MB, 11-way set associative, non-inclusive write-back
DRAM	six 16-GB DDR4 DIMMs × 2 sockets
NVM	six 128-GB Optane DC NVDIMMs × 2 sockets
Interconnect	Intel® UPI at 10.4 GT/s, 10.4GT/s, and 9.6 GT/s

Table II: Evaluated benchmarks.

Benchmark	**Input Problems**
Hypre [34]	a 3D electromagnetic diffusion problem
Laghos [7]	the Sedov blast wave Q3-Q2 3D computation
ScaLAPACK [4]	the distributed matrix multiplication of dimension NxN
NPB [2] - FT	a discrete 3D fast Fourier Transform of class D
HACC [10]	a 252 simulation box using 384 grids in CORAL benchmark suite
BoxLib (AMReX) [3]	the spherical chemical wave propagation
XSBench [27]	the unionized grid of XL problem with 34 million lookups
SuperLU [17]	a distributed PDGSSVX routine with real datasets from [6]

IV. Performance Analysis

This section evaluates the impact of NVM-based main memory on HPC scientific applications from three aspects – (1) the performance sensitivity in cached-and uncached-NVM (2) the write throttling effect on critical phases (3) the diverging effect of concurrency changes on read and write accesses. We also quantify the performance impact of checkpointing on NVM.

A. Overall Performance

We start the evaluation with an overview of performance sensitivity of HPC applications on two NVM-based main memory, i.e., cached and uncached. All the experiments use the local socket to eliminate the severe NUMA effects reported in [9, 12, 21]. Input problems have a memory footprint fit in DRAM capacity (50%-85%) so that we can use the performance on DRAM-only main memory as the reference. Figure 2 reports application performance on DRAM-only, cached-NVM, and uncached-NVM main memory. Note that SuperLU, XSBench, and FFT use application-defined metrics, i.e., the higher, the better, while the other applications report the run time, i.e., the lower, the better.

All applications on the cached-NVM manage to achieve performance comparable to that on the DRAM. The performance gap between DRAM and the cached-NVM is less than 10% except for ScaLAPACK, Hyre, and BoxLib.

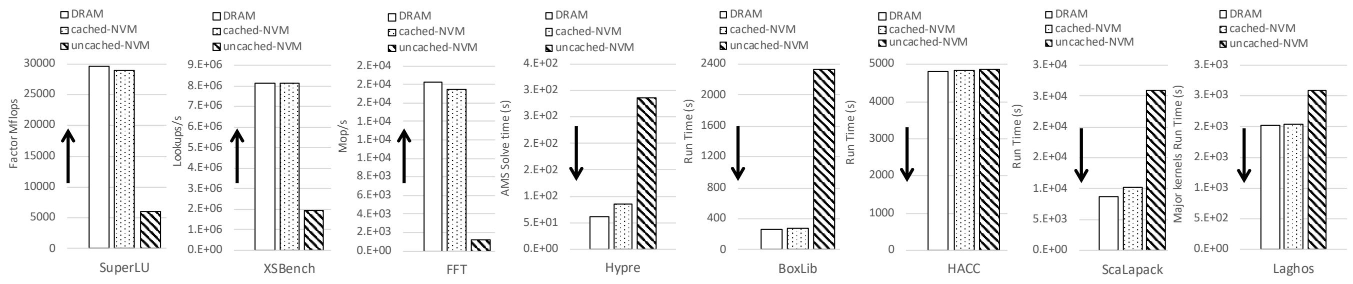


Figure 2: An overview of the performance sensitivity of eight applications to cached and uncached NVM compared to DRAM. SuperLU, XSBench, and FFT report application-defined performance metrics and the others report the run time.

Table III: An overall characterization of performance sensitivity to NVM-uncached. The last column classifies applications into three tiers. Highlighted cells in the same color are the primary indicator for each tier.

Dwarf	**Application**	**Memory BW (MB/s)**	**Read BW (MB/s)**	**Write BW (MB/s)**	**Write Ratio(%)**	**Slowdown(x)**
N-body	HACC	40	25	14	36	1.01
Structured Grid	Lagos	4,135	3,114	1,021	25	1.27
Dense Linear Algebra	Scalapack	11,984	10,104	1,880	16	2.99
Monte Carlo	XSBench	16,134	16,130	4	0.03	4.16
Structured Grids	Hypre	11,413	10,519	894	8	4.67
Sparse Linear Algebra	SuperLU	12,441	11,240	1,201	10	4.94
Unstructured Grids	BoxLib	10,334	8,246	2,088	21	8.94
Spectral Methods	FFT	6,365	4,063	2,302	36	14.92

These three applications have more performance loss, with a maximum loss of 27% in Hypre (to be analyzed in the next section). Note that cached-NVM requires no porting efforts from the application developer, which would likely be the first deployment choice.

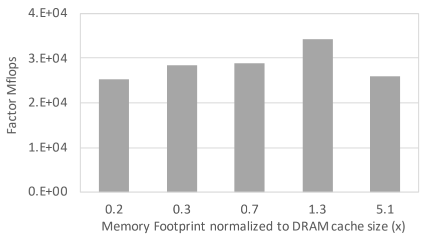
On the uncached NVM, applications exhibit three tiers of performance sensitivity, i.e., *insensitive*, *scaled*, and *bottlenecked* performance compared to the DRAM baseline. We group applications into three groups and summarize the profiling results of memory traffic in Table III. In the first tier, HACC and Laghos show insignificant performance difference when the main memory switches from DRAM to NVM directly. The loss in performance is less than the difference in latency and bandwidth of NVM and DRAM. This class of applications is characterized by low memory bandwidth (in green). Scientific applications that share similar computations as HACC (N-body) and Laghos (unstructured finite element) may sustain performance when porting onto the NVM-based main memory without a DRAM cache.

Four applications form in the second tier, showing scaled performance on the uncached NVM, as compared to their DRAM baseline. These applications exhibit 2.99 to 4.94 times slowdown, which approximates the three times performance gap between DRAM and NVM, as benchmarked on the testbed [21]. We characterize this tier of applications by a high memory bandwidth but a low write ratio. The total memory bandwidth ranges between 11 and 16 GB/s while their write bandwidth remains lower than 2 GB/s, making up 0.03% to 16% total bandwidth only. Finally, BoxLib and FFT form the third tier of performance sensitivity. Their performance is severely bottlenecked on the uncached NVM, showing a slowdown that is much higher than the latency and bandwidth difference in the two memories. Although this group of applications has a total memory bandwidth lower than applications in the second tier, they have higher than 2 GB/s write traffic. In particular, their memory traffic has read/write ratios as low as 1.5 so that the write traffic could even reach 36% total memory traffic. We notice that the peak write bandwidth to NVM could reach 12 GB/s on the testbed, which indicates the application slowdown is not a result of bandwidth saturation. We identify the write throttling effect and concurrency contention as the primary causes of the slowdown in Section IV-D.

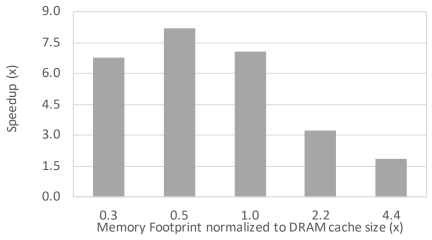
Observation I: N-body and structured grids applications may have indifferent or scaled changes on NVM main memory while spectral methods may have severe degradation.

B. Cache Efficiency

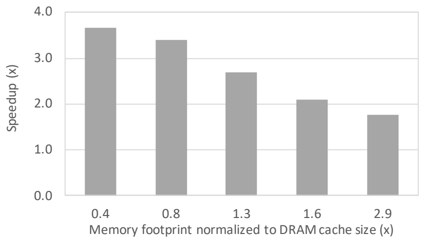
We use two metrics to quantify the effectiveness of using DRAM as a cached to NVM. First, for input problems with memory footprint smaller than the DRAM capacity, we define *cache efficiency* as the relative performance to that on DRAM directly. The expectation is that low hardware overhead for managing DRAM as a cache should bring the performance to match using DRAM natively. Second, when the input problems require memory size more than the DRAM capacity, we defined *cached speedup* as the performance improvement from non-cached NVM. In this scenario, the cached-NVM is expected to enable larger problems than on DRAM while still sustaining higher per-



(a) SuperLU.



(b) BoxLib.



(c) Hypre.

Figure 3: The performance of SuperLU, BoxLib, and Hypre on the cached-NVM at increased input problems. The x-axis reports the memory footprint normalized to the capacity of the DRAM cache. SuperLU reports the application-defined metric while BoxLib and Hypre report the speedup in execution time on cached-NVM.

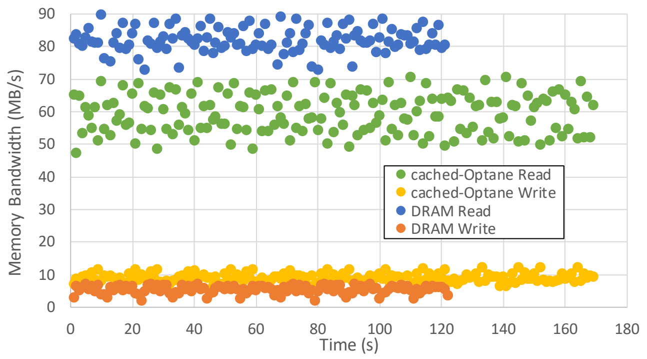


Figure 4: The trace of read and write bandwidth when Hypre run on cached-NVM and DRAM respectively.

formance than uncached-NVM.

Overall, cached-NVM delivers high cache efficiency, as shown in Figure 2. Among them, Hypre has the lowest cache efficiency with a 28% performance loss. We identify the cause by collecting samples of read and write bandwidth to each DRAM DIMM throughout the execution. The reconstructed traces of memory traffic in the cached-NVM and DRAM configurations are presented in Figure 4. When Hypre executes in cached-NVM, it has an average write bandwidth at 9.3 GB/s. The write bandwidth drops nearly by half when it runs on DRAM mode, with an average of 5.7 GB/s. We believe that this increased write bandwidth in cached-NVM is the cause for the decrease in the read bandwidth. The read bandwidth decreases from 82.5 GB/s in DRAM mode to 59.5 GB/s in cached-NVM, i.e., an exact 28% reduction that matches the performance loss. Since Hypre is a read-dominant workload, the read bandwidth directly impacts the overall performance.

We evaluate the second metric by scaling up the input problems to observe changes in application performance. We focus on three MPI applications, i.e., SuperLU, BoxLib, and Hypre, which typically require multiple compute nodes on supercomputers for realistic simulations. For SuperLU, we use five real datasets (kim2, offshore, Ge87H76, nlpkkt80, and nlpkkt120) from [6], where the largest input requires 490 GB memory. For BoxLib and Hypre, we scale up their simulation domains to reach 300 GB memory footprint. We report the application-defined metric in SuperLU and the speedup in runtime in BoxLib and Hypre in Figure 3. SuperLU sustains similar performance when the input problems scale up to five times DRAM capacity. Both BoxLib and Hypre show decreased speedup when their input problems increase. When the memory footprint is about 4.4 and 2.9 times the DRAM capacity, the cached-NVM still manages to double the performance compared to uncached mode. Overall, these applications that represent the sparse linear algebra and structured grid Dwarfs may benefit from the hardware managed cache to NVM for efficient execution of large-scale problems.

Observation II: sparse linear algebra and structured/unstructured grids applications on cached-NVM could achieve improved performance even for problems substantially beyond DRAM capacity.

C. *Write Throttling*

Uncached-NVM exposes the characteristics of NVM directly to the application without the interference from DRAM cache. We analyze application performance in this mode to provide feedback and insights for future NVM-based designs. Asymmetric read and write performance is a common characteristic of NVM technologies. On the testbed, the asymmetry is quantified as 39 GB/s read bandwidth and 13 GB/s write bandwidth. Our analysis reveals a *write-throttling effect* that could change the critical computation phase of an HPC application when running on uncached-NVM.

SuperLU and Laghos both have two distinct phases in the execution, as shown in the trace of memory traffic in Figure 5. These phases, however, show different sensitivity to the write throttling effect. The first phase in Laghos always takes about 20% the execution time when running on DRAM (Figure 5a) and uncached-NVM (Figure 5b). In contrast, when running on DRAM, the first phase in SuperLU only takes 20% the execution time (Figure 5c), but

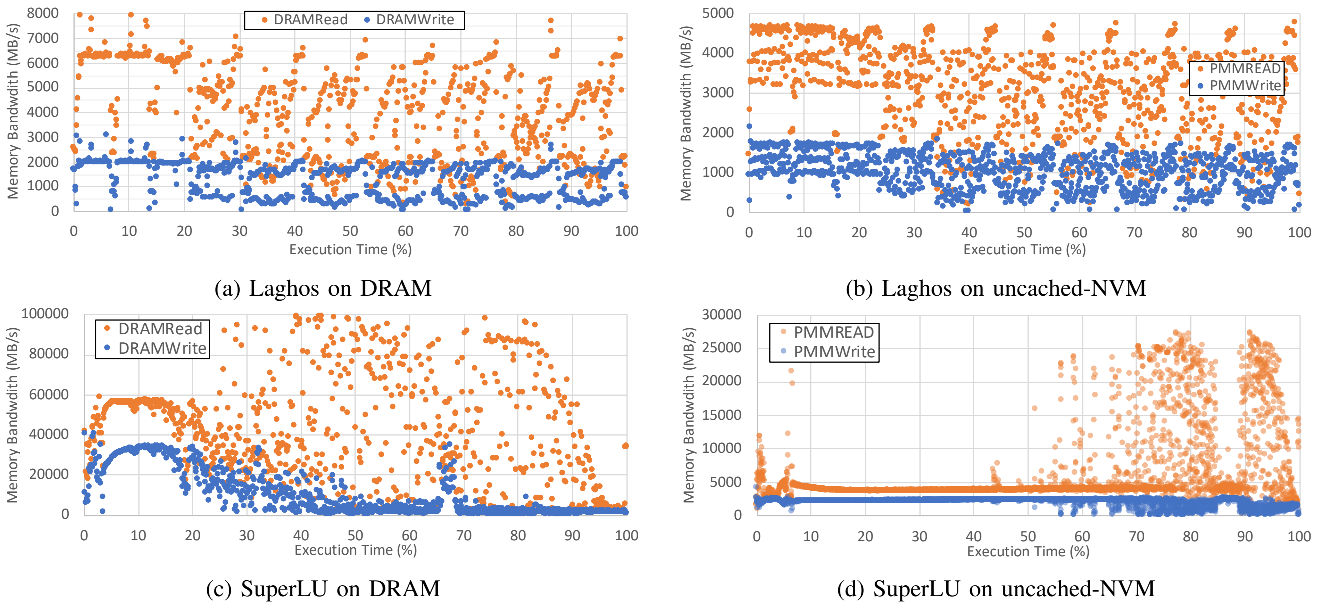


Figure 5: Write throttling changes the dominant computation phase in SuperLU from 20% to 70% execution time. In contrast, Laghos sustains similar composition of computation phases in the two memory modes.

significantly extends to 60% execution time on uncached-NVM (Figure 5d).

We find that there exists a threshold value of the write bandwidth, above which a computation phase will significantly extend the execution. We quantify this threshold as 2 GB/s on the testbed. For instance, the first phase in Laghos has a moving average of 1.3 GB/s write bandwidth with its peak remaining lower than 2 GB/s when running on DRAM. The read/write ratio remains at 3 in this stage. These characteristics are unchanged when Laghos runs on uncached-NVM. On the other hand, the first phase in SuperLU exhibits high write traffic and low read/write ratio when running on DRAM, resulting in an average of 33 GB/s and a peak at 40 GB/s. When running on uncached-NVM, the write bandwidth of this phase is reduced by 14 times, reaching only 2.3 GB/s. The dramatically reduced write performance *throttles* the read performance due to data dependency and coupling effects in shared units [20]. Consequently, the read performance is also reduced significantly from 54 GB/s to 4 GB/s. It is a high priority to address this change of behavior in critical phases when optimizing applications on NVM-based main memory.

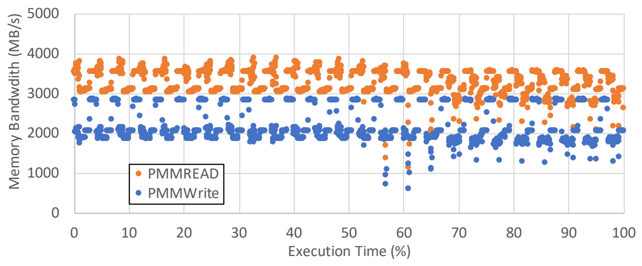
We identify low read/write ratio and high write bandwidth as the indicator to detect applications that are susceptible to the write throttling effect. Comparing the two phases in SuperLU, we find that the second phase with a high read/write ratio and low write traffic has only a 'scaled' slowdown, which is proportional to the performance gap between DRAM and NVM. In Laghos, read and bandwidth on DRAM in the two phases is lower than the peak bandwidth of NVM, and thus the changes in application performance are minimal. We highlight that phase-specific characteristics become increasingly important in identifying the bottleneck of HPC applications on NVM because the throttling effect could dramatically change the profile of execution time.

Observation III: HPC applications with computation phases susceptible to the write throttling on uncached-NVM require different priorities in optimization.

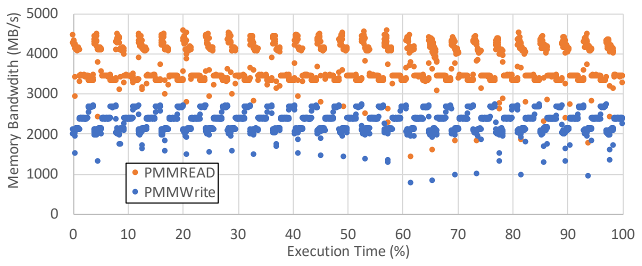
D. Concurrency Contention

HPC applications on supercomputers often exploit the high parallelism from multicore processors on multiple nodes to accelerate simulations. However, multiple threads may contend on shared buffers or units in the memory, creating a performance bottleneck. For instance, re-ordering and merging write to NVM is a common technique to mitigate the high energy cost and low write bandwidth of NVM technologies. On the testbed, write pending queues (WPQ) in the memory controller functions for this purpose. If the concurrency is high, contention may arise on a fully occupied WPQ when new requests have to wait for the WPQ to drain before being inserted into the queue [32]. Also, high concurrency would decrease the opportunity of combinable requests in WPQ, similar to the well-known fact that high concurrency reduces the locality in the row buffer.

We identify concurrency contention by comparing the performance change at different concurrency across memory configurations. Each application is executed at a low and a high concurrency on DRAM, cached-NVM, and uncached-NVM, respectively. The contention ratio is calculated as the performance at the high concurrency normalized to that at the low concurrency. Figure 7 reports the ratios in the



(a) $Concurrency = 8$



(b) $Concurrency = 24$

Figure 6: A diverging impact from concurrency changes in read and write in FT. The reduced write bandwidth overpowers the increased read bandwidth, resulting a 26% performance loss.

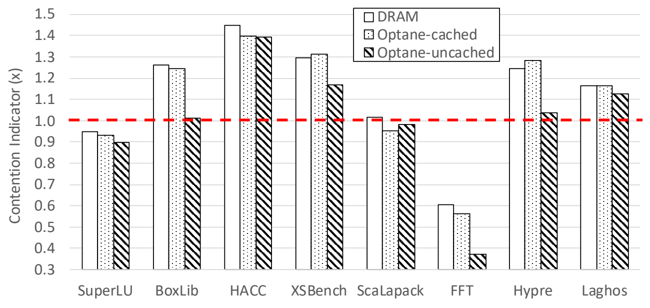
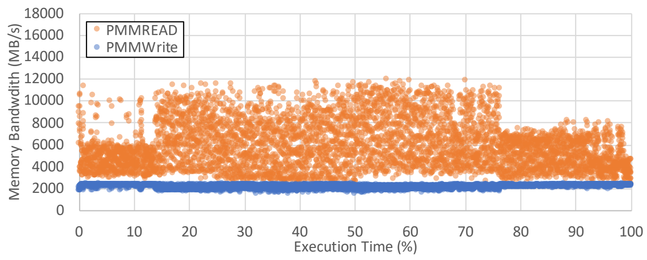
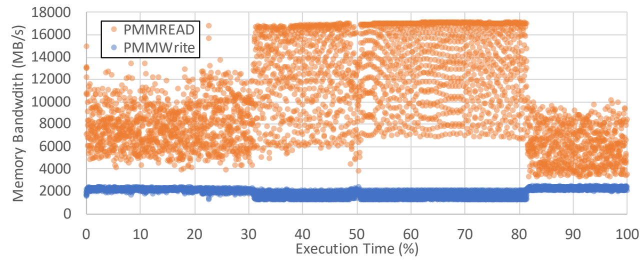


Figure 7: Applications with contention ratio lower than the red dotted line show performance loss when the level of concurrency increases. Larger gap between DRAM and uncached-Optane indicates more contention on NVM.



(a) $Concurrency = 16$



(b) $Concurrency = 36$

Figure 8: Increased concurrency in ScaLAPACK prolongs the first stage (the left of the dashed line) from 10% execution time to 30%, and also increases the gap between read and write bandwidth in the first 80% execution time.

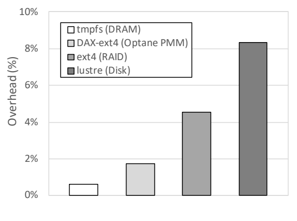
eight applications. Applications with a ratio larger than one (the red dotted line) benefit from increased concurrency. For instance, HACC and XSBench have more than 30% performance improvement when their concurrency increases. A ratio lower than one may not necessarily be a result of contention on memory. Some algorithmic properties could also cause reduced performance. Thus, we use the difference between ratios on DRAM and other configurations to identify the concurrency contention. For instance, FFT has a ratio of 0.61 on DRAM but only 0.37 on uncached-NVM, indicating the contention from NVDIMM as the main cause for performance loss. Interestingly, ScaLAPACK has higher contention on cached-NVM than uncached-NVM.

We reconstruct the trace of memory traffic of FT and ScaLAPACK at two concurrency levels in Figure 6 and 8. FT consists of iterative phases. In each phase, the write bandwidth at the lower concurrency can reach 3 GB/s (6a) while at the high concurrency it is below 2.7 GB/s. The increased concurrency, however, has an opposing effect on read bandwidth, which increases from 3.8 GB/s to 4.5 GB/s. Overall, increased concurrency increases the divergence between the read and write bandwidth. This conflicting effect could also change the composition of computation phases. In ScaLAPACK, the first stage extends from 10% execution time in Figure 8a to 30% in Figure 8b. Note that read bandwidth in the second stage increases dramatically from 12 GB/s to 17 GB/s, i.e., a reduced execution time. Since the execution time of the first stage remains unchanged, its portion in the total execution increases.

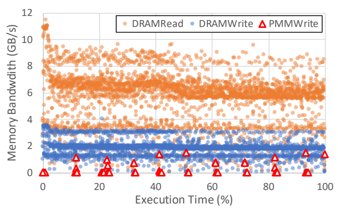
Observation IV: Concurrency changes may have a diverging impact on read and write bandwidth. Phase-specific optimization or write-aware data placement may be more effective than a global adjustment of concurrency.

E. Leveraging the memory persistence

Large-scale HPC simulations often rely on I/O intensive visualization and checkpointing to detect anomaly at an early stage and ensure the completion of long-running jobs. When Optane is used as a persistent memory, HPC applications



(a) Overhead on four tiers of storage.



(b) The trace of bandwidth.

Figure 9: Visualization in Laghos snapshots every five steps.

may directly benefit from its high bandwidth and memory persistence. We configure Optane in App Direct mode and evaluate the overhead of visualization in Laghos on four tiers of storage, from tmpfs on DRAM, a DAX-aware ext4 file system on the Optane, an ext4 file system on the local RAID, to a Lustre file system on network interconnected disk. Note that tmpfs is not persistent but provides the upper bound of performance. The results are consistent with the memory/storage hierarchy, as shown in Figure 9a. The Optane persistent memory only imposes 2%-5% overhead, achieving four times reduction in overhead on other persistent storages. We further analyze the interaction between PMM and DRAM traffic in Figure 9b. The bandwidth to PMM is periodic and write-only. There is no interference to the traffic to DRAM throughout the execution.

V. Performance Optimization

In this section, we propose two optimization techniques for cached- and uncached-NVM main memory, respectively. We develop a prediction model to predict performance changes in cached-NVM when the concurrency or data size changes. On uncached-NVM, we employ explicit data placement to improve performance with reduced data size on DRAM.

A. Model-based Prediction

Our performance analysis of applications on cache-NVM reveals that changes in concurrency level and data size both can impact the effectiveness of execution significantly. Naturally, if a performance model can predict application performance at various configurations, it could help the application developer select an optimal set up without exhaustively search all possible configuration space. Suppose a general configuration consists of multiple dimensions of freedom. Our empirical observation indicates that when sweeping the configuration in one dimension, performance impact may change from positive to negative, which will reflect in a similar trend in some hardware events. These events are defined as critical events and used as predictors [5].

A set of critical events mutually indicate the overall trend in the performance of an application at a specific configuration. This behavior is modeled analytically in a multivariate function (Eq. 1). Here, each variant N_{e_i} represents the count of one critical event e_i. β_i, the coefficient, indicates either a positive or negative impact on the derivative of performance. Our selection of critical events combines empirical observation and a statistical procedure. First, from the classification of performance sensitivity to NVM main memory, we identify several critical indicators, such as the computation intensity, memory traffic, read and write ratios. These metrics could be reflected in a range of hardware events. Second, we test a set of relevant hardware events into the regression model to prune highly correlated events, i.e., high p-values. Table IV summarize the events selected for deriving the prediction model.

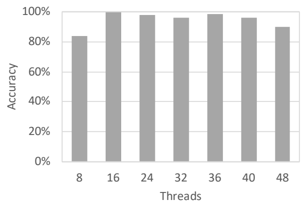
$$IPC_p = \sum_{n=1}^{N} \beta_i \cdot (N_{e_i} \cdot IPC_s) + \sigma \qquad (1)$$

Table IV: The events selected for performance prediction.

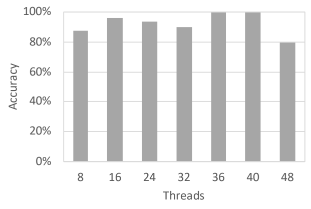
Feature	Activities
p_0	Instruction Retired
p_1	Cycles Active
p_2	Cycles stalled due to Resource Related reason
p_3	Cycles in waiting for outstanding offcore requests
p_4	Count of the number of reads issued to memory controllers.
p_5	Counts of Writes Issued to the iMC by the HA.

We use two data collection strategies to collect training data sets for models for concurrency and data size separately. When deriving the model for predicting performance at different concurrency, we collect hardware events from application executions at the *middle point* concurrency. For instance, for hardware with HT hardware concurrency, we collect data sets from executions using $0.75HT$. When deriving the model for predicting performance at a different data size, we fix the concurrency and collect events from configurations at a small data size. Next, the measurement for each hard event is first scaled by the sampled IPC (IPC_s in Eq. 1) and then normalized by calculating their zero scores. The normalized features are used as the training data set to derive the coefficients of Eq. 1 using multivariate linear regression.

We evaluate the accuracy of the prediction by comparing the estimated IPC with the observed IPC. In the first experiment, we predict the application performance at different concurrency. We collect training data sets from running applications in the configuration of $ht = 36$ only. The prediction model is derived from this training data set to estimate the performance at other concurrency levels. The estimation error E_{est} is calculated as the absolute difference of prediction and observed divided by the observed IPC. In Figure 10, we report the accuracy as $1 - E_{est}$ for XSBench and FT, respectively. The average prediction errors in the two applications are 5% and 8%. All concurrency levels except the lowest and highest level have accuracy above 90%. In the second experiment, we derive the prediction for various data sizes at concurrency $ht = 36$. We collect the training data set from each application execution using a small input problem. The derived model then predicts the application performance at larger input problems. Figure 11 reports the prediction accuracy when the data size (x-axis) increases in XSBench and ScaLAPACK. While all data sizes in ScaLAPACK have accuracy over 97%, XSBench has lower accuracy at the largest data size.

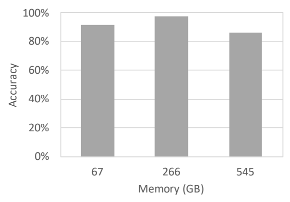


(a) $XSBench$

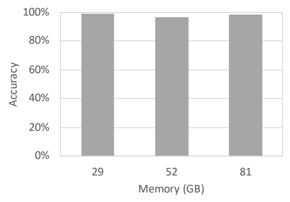


(b) $NPB-FT$

Figure 10: The modeling accuracy of concurrency change.



(a) $XSBench$



(b) $ScaLAPACK$

Figure 11: The modeling accuracy of data size change.

B. Write-aware Data Placement

In the second optimization technique, we target uncached-NVM in heterogeneous memory. From the performance analysis, we show that some applications like ScaLAPACK and FT in Section IV-D could have diverging effects on read and write bandwidth from the changes in concurrency. Instead of adapting the concurrency, we explore a different optimization technique that employs write-aware data placement in applications to bypass this effect. This optimization places data structures with substantial write traffic onto DRAM so that increased concurrency would still increase read bandwidth from NVM but would not create contention due to writes on NVDIMMs.

We use a hardware sampling-based implementation of the data-centric profiling tool [22] to identify write-intensive memory objects for placement. The application source is then modified accordingly to place the write-intensive data structures onto DRAM using APIs in [21]. Figure 12 presents the performance of the optimization as compared to that on DRAM and uncached-NVM, respectively. At different data sizes, the optimization manages to achieve DRAM-similar performance. Note that the used DRAM capacity in the optimized implementation is only 30% of the DRAM and cached-NVM modes. As a validation, we also test placing other read-intensive memory objects onto DRAM, and the performance shows minimal changes, which approximates that on uncached-NVM.

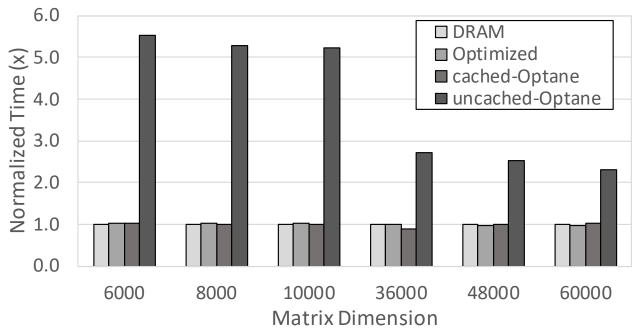


Figure 12: Optimization on Scalapack by changing data placement on the NVM-based heterogeneous system.

VI. Related Work

Before the hardware of byte-addressable NVM becomes available, most previous works use emulators and simulators for evaluating their approaches on NVM [8, 13, 19, 20, 23, 29, 30, 31, 33]. Although simulations and emulations can provide valuable insights into the performance trend, they either lack the performance details or are constraint by small problems due to the long simulation time. In [12] and [32], the authors prove that using software emulation or hardware emulation does not capture all the features of real hardware like the Intel Optane. Therefore, the system software for NVM proposed in the previous studies requires re-evaluation. Different from these works, the findings and insights in this work are derived from representative HPC applications on real NVM hardware.

Since the release of real hardware, several works have provided preliminary evaluations of the Intel Optane DC PMM [12, 21]. Some works have also ported selected applications in commercial database, scientific and graph workloads onto Optane [9, 18, 28, 32]. For instance, [9] optimizes the graph workload Galois to mitigate the NUMA effect when Optane is in the Memory mode. Still, a comprehensive evaluation that covers the landscape of HPC applications as in our work is missing.

Prior works have proposed various approaches for utilizing NVM-based heterogeneous memory systems [8, 20, 30, 31]. Unimem [30] uses a sample-based approach to collect memory access information to decide data placement on NVM-DRAM systems. Siena [20] explores different organizations and configurations of DRAM and NVM in a memory system to decide optimal system designs for HPC applications. NVStream [8] utilizes the byte-addressability in NVM to remove expensive system calls and uses non-temporary storage and delta compression to reduce overhead due to ensuring crash consistency on NVM. In this work, we identify new optimization priorities and insights that will also benefit these approaches and techniques.

VII. Conclusion

In this work, we analyze the performance of HPC applications representative for the Seven Dwarfs on NVM-based heterogeneous memory hardware. Our results quantify the effectiveness of using DRAM as a cache to NVM to improve performance at large input problems. For uncached NVM, we identify that the write throttling effect and concurrency contention demands a high priority of optimization. We highlight that changing concurrency could have diverging impacts on read and write bandwidth in some applications. Therefore, a global adjustment of the concurrency may be insufficient. For the cached-NVM, we develop a prediction model based on hardware events collected from a small set of application runs to predict the performance at various concurrency and data size. For uncached-NVM, we demonstrate that write-aware data placement can effectively accelerate applications with reduced requirement of DRAM capacity. Our results show that the new memory system hardware is promising for future supercomputer designs.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. 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. LLNL-PROC-793886. This work was partially supported by U.S.National Science Foundation (CNS-1617967, CCF-1553645 and CCF-1718194).

References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California at Berkeley, 2006.

[2] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

[3] J Bell, A Almgren, V Beckner, M Day, M Lijewski, A Nonaka, and W Zhang. Boxlib users guide. *github. com/BoxLib-Codes/BoxLib*, 2012.

[4] L Susan Blackford, Jaeyoung Choi, Andy Cleary, Eduardo D'Azevedo, James Demmel, Inderjit Dhillon, Jack Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, et al. *ScaLAPACK users' guide*, volume 4. Siam, 1997.

[5] Matthew Curtis-Maury, Filip Blagojevic, Christos D Antonopoulos, and Dimitrios S Nikolopoulos. Prediction-based power-performance adaptation of multithreaded scientific codes. *IEEE Transactions on Parallel and Distributed Systems*, 19(10):1396–1410, 2008.

[6] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[7] Veselin A Dobrev, Tzanio V Kolev, and Robert N Rieben. High-order curvilinear finite element methods for lagrangian hydrodynamics. *SIAM Journal on Scientific Computing*, 34(5):B606–B641, 2012.

[8] Pradeep Fernando, Ada Gavrilovska, Sudarsun Kannan, and Greg Eisenhauer. Nvstream: Accelerating hpc workflows with nvram-based transport for streaming objects. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, 2018.

[9] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. Single machine graph analytics on massive datasets using intel optane dc persistent memory, 2019.

[10] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. HACC: extreme scaling and performance across diverse architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 6. ACM, 2013.

[11] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 459–462. IEEE, 2005.

[12] 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. Basic performance measurements of the intel optane dc persistent memory module, 2019.

[13] Aasheesh Kolli, Steven Pelley, Ali Saidi, Peter M. Chen, and Thomas F. Wenisch. High-Performance Transactions for Persistent Memories. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[14] Argonne National Lab. 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.

[15] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. *ACM SIGARCH Computer Architecture News*, 37(3):2–13, 2009.

[16] Dong Li, Jeffrey Vetter, Gabriel Marin, Collin McCurdy, Cristi Cira, Zhuo Liu, and Weikuan Yu. Identifying Opportunities for Byte-Addressable Non-Volatile Memory in Extreme-Scale Scientific Applications. In *International Parallel and Distributed Processing Symposium*, 2012.

[17] Xiaoye S Li. An overview of superlu: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):302–325, 2005.

[18] Onkar Patil, Latchesar Ionkov, Jason Lee Lee, Frank Mueller, and Michael Lang 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 '19, 2019.

[19] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. Memory Persistency. In *ISCA*, 2014.

[20] I. B. Peng and J. S. Vetter. Siena: Exploring the design space of heterogeneous memory systems. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 427–440, Nov 2018.

[21] Ivy B. Peng, Maya B. Gokhale, and Eric W. Green. System evaluation of the intel optane byte-addressable NVM. In *Proceedings of the International Symposium on Memory Systems*. ACM, 2019.

[22] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Pietro Cicotti, Erwin Laure, and Stefano Markidis. Rthms: A tool for data placement on hybrid memory system. In *ACM SIGPLAN Notices*, volume 52, pages 82–91. ACM, 2017.

[23] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Jeffrey S Vetter, Pietro Cicotti, Erwin Laure, and Stefano Markidis. Characterizing the performance benefit of hybrid memory system for hpc applications. *Parallel Computing*, 76:57–69, 2018.

[24] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 24–33, New York, NY, USA, 2009. ACM.

[25] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80, 2008.

[26] Thomas Willhalm, Roman Dementiev, Patrick Fay. Intel performance counter monitor - a better way to measure cpu utilization, 2017.

[27] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

[28] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory I/O primitives. In *Proceedings of the 15th International Workshop on Data Management on New Hardware*, DaMoN'19, 2019.

[29] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2011.

[30] Kai Wu, Yingchao Huang, and Dong Li. Unimem: Runtime Data Management on Non-volatile Memory-based Heterogeneous Main Memory. In *SC*, 2017.

[31] Kai Wu, Jie Ren, and Dong 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*, 2018.

[32] Jianhua Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. An empirical guide to the behavior and use of scalable persistent memory. *ArXiv*, abs/1908.03583, 2019.

[33] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and

Bingsheng He. NV-Tree: Reducing consistency cost for nvm-based single level systems. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015.

[34] Ulrike Meier Yang. Parallel algebraic multigrid methodshigh performance preconditioners. In *Numerical solution of partial differential equations on parallel computers*, pages 209–236. Springer, 2006.