Siena: Exploring the Design Space of Heterogeneous Memory Systems

Ivy B. Peng
Future Technologies Group
Oak Ridge National Laboratory
Oak Ridge, USA
pengb@ornl.gov

Jeffrey S. Vetter
Future Technologies Group
Oak Ridge National Laboratory
Oak Ridge, USA
vetter@computer.org

***Abstract*—Memory systems are crucial to the performance, power, and cost of high-performance computing systems. Recently, multiple factors are driving the need for more complex, deep memory hierarchies. However, architects and customers are struggling to design memory systems that effectively balance multiple, often competing, factors in this large, multidimensional, and fast-moving design space. In this paper, we systematically explore the organization of heterogeneous memory systems on a framework called Siena. Siena facilitates quick exploration of memory architectures with flexible configurations of memory systems and realistic memory workloads. We perform a design space exploration on 22 proposed memory systems using eight relevant workloads. Our results show that horizontal organizations of memories can achieve higher performance than vertical organizations when the distribution of memory traffic balances the performance gap between memories. However, the coupling effects through shared resources and application behaviors could negate the advantage of high-performance memory in horizontal organizations.**

***Index Terms*—Heterogeneous Memory Systems, System Exploration, Design Space Exploration**

I. Introduction

Memory systems play a critical part in high-performance computing (HPC) systems, directly impacting their performance, power consumption, and cost. Recent requirements from applications and constraints from technologies are shifting HPC systems towards more complex, deep memory hierarchies. First, requirements for memory capacity continue to increase dramatically in response to simulation, machine learning, and enterprise applications [17], [42]. Second, the continued imbalance of computing performance to memory performance constricts overall application performance. Third, in HPC specifically, the plateauing performance of both I/O subsystems and interconnection networks is forcing applications to consider alternative scenarios like in situ visualization and analytics that utilize large on-node memory to reduce expensive inter-node data movement. Finally, the conventional memory technology – dynamic random-access memory (DRAM) – is approaching its limits in terms of density, power, and cost; hence, researchers are investigating new technology options [4], [26], [47]: non-volatile memory (NVM), such as 3D-XPoint [14], phase-change memory (PCM [27]), 3D NAND flash [33], and spin-transfer-torque magnetic RAM (STT-MRAM [51]). Meanwhile, high-performance volatile memory, like Hybrid Memory Cube (HMC [7]), Wide-I/O 2 (WIO2 [18]), high-bandwidth memory (HBM [19]), and GDDR6 [20], continues to be actively developed and deployed.

To address these concerns, computer architects have turned to heterogeneous memory systems as a solution to balance their designs. Not surprisingly, this strategy provides considerably more flexibility, but it also presents new risks and challenges. First, this large, multidimensional design space is expensive to evaluate with existing tools for functional or cycle-accurate simulations. Additionally, applications, code-generation toolchains, and other architectural components must adapt to each memory technology under consideration. For example, in a horizontal scratchpad organization, applications must be explicitly ported to use these new memory features as they are not transparently managed by the hardware or the operating system. Second, these new memory systems must be designed for a specific set of applications (*application-specific*) in terms of computational intensity, working-set size, memory access patterns, parallelism, etc. If the specific application requirements and scaling predictions are inaccurate, then the ultimate design could miss the optimal balance. Third, in this period of expanding options, many of these memory systems will be unique in that they are immature and the first implementation of this memory architecture. Designer experience, existing toolchains, and performance estimates will be less obtainable and require more efforts.

As a result, architects and customers are struggling to design HPC memory systems that effectively balance multiple factors of cost, performance, capacity, and power. Moreover, many diverse technologies, such as NAND flash, HBM2, GDDR6, are being rapidly improved [5], [6], [24] and must be evaluated frequently as new parameters become available. In this regard, efficient and flexible design tools for memory

This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. This research was supported in part 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. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the U.S. Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00 ©2018 IEEE

systems for analyzing and optimizing these options are gaining in importance.

A. Contributions

In this paper, we address the challenge of exploring the state of the current design space of heterogeneous memory systems using an innovative framework called *Siena*. *Siena* is designed to facilitate quick exploration of diverse memory architectures. We use *Siena* to identify promising design options by comparing the relative performance of system options using application-specific workloads. We summarize our contributions as follows:

- We survey the design space of emerging memory technologies and potential memory architectures for HPC nodes.
- We provide *Siena* framework to facilitate the memory system exploration. *Siena* uses abstract, scalable application models to drive memory simulations of flexible organizations of heterogeneous memory systems.
- We perform a comprehensive design space exploration of 22 proposed memory systems, including vertical and horizontal organizations of memories.
- We analyze the utilization of the high-performance memory in vertical organizations. We show that vertical organizations are preferred for workloads that can sustain the utilization on different problem sizes.
- We reveal the coupling effects among memories in horizontal organizations. We show that shared resources and application characteristics could result in back pressure from slow memory, which hides the advantage of high-performance memory.

The rest of this paper is organized as follows. We explain the two exploration directions in §II and introduce our exploration approach in §III. We describe the experimental methodology in §IV. We analyze the exploration results and draw insights to system designs in §V. In §VI, we discuss other related works. Finally, we conclude our work in §VII.

II. Design Space of Emerging Memory Systems

In recent years, memory systems have become much more complex. As computer architects balance competing objectives, the emerging memory system designs have many more devices and architectural dimensions than the cache-based memory hierarchies of SRAM and DRAM in the past two decades. As shown in Table I, various new devices, such as 3D-XPoint and HMC, provide new capabilities for power efficiency, density, capacity, and performance. Organizing these memory devices into one memory system is also exposed to an extensive exploration space (Figure 2). Meanwhile, memory system architectures must be designed for application-specific workloads, costs, and reliability, and the current design options are quite broad.

TABLE I: Comparison of four tiers of memory technologies [5], [7], [13], [14], [18]–[21], [24], [26], [32], [39], [40], [48], [50], [51].

	Volatile	Density (GB)	BW (GB/s)	Est. Cost	Speed	Latency
HMC Gen2	✓	4-8	320	3x	30 Gbps	$\sim$100s ns
HBM2	✓	2-8	256	2x	2 Gbps	$\sim$100s ns
GDDR6	✓	1-2	72	2x	18 Gbps	$\sim$100s ns
WIO2	✓	1-4	68	2x	1,066 MT/s	$\sim$100s ns
DDR4	✓	0.25-2	25.6	1x	3,200 MT/s	20-50 ns
STT-MRAM	✗	0.25	10.6	1x	1,333 MT/s	10-50 ns
PCM	✗	1	3.5	1x	3M IOPS	50-100 ns
3D-XPoint	✗	750	2.4	0.5x	550K IOPS	10 μs
Z-NAND	✗	800	3.2	0.5x	750K IOPS	12-20 μs
NAND Flash	✗	>1,000	<3	0.1x	50K IOPS	25-125 μs

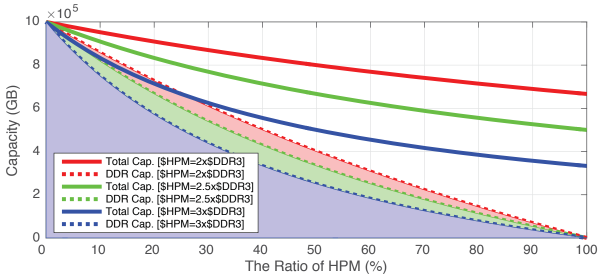


Fig. 1: Impact of the high-performance memory price on the total capacity of a memory system under a fixed budget.

The objective of this work is to employ a systematic approach for exploring designs of heterogeneous memory systems. A feasible memory configuration has constraints on cost, including both hardware cost and power consumption. It also needs to provide sufficient capacity and reasonable performance. We model a heterogeneous memory system as composed of N different memory technologies, denoted by M_i $(i = 1..N)$. Under a fixed budget for a memory system, we demonstrate in Figure 1 an example of using two memories: a DDR3 SDRAM and a high-performance memory (HPM). When the price or capacity of HPM increases, the total memory capacity (solid lines) decreases. Our exploration starts from the basic requirements from applications, i.e., the minimum capacity for enabling typical workloads. We assume that a HPM has a higher unit price than a conventional memory (see the estimated cost in the fourth column of Table I). Therefore, we try to minimize the ratio of HPM in a memory system without causing significant performance degradation. Our approach provides a framework to explore various memory configurations under application-specific workloads. In the remainder of this section, we introduce the state-of-art memory technologies and organizations.

A. Memory Technologies

Heterogeneous memory systems are emerging as the dominant memory technology, DRAM, faces challenges in scaling. Although various memory technologies are being actively developed, none of them can fully replace DRAM. Composing a heterogeneous memory system can potentially take advantage of multiple memory technologies.We summarize promising technology options in Table I that consists of four tiers.

HPM is the top tier of memory technologies and is represented by HMC [7], HBM [19], Wide I/O2 [18], and GDDR6 [20]. These technologies can provide over 10 times

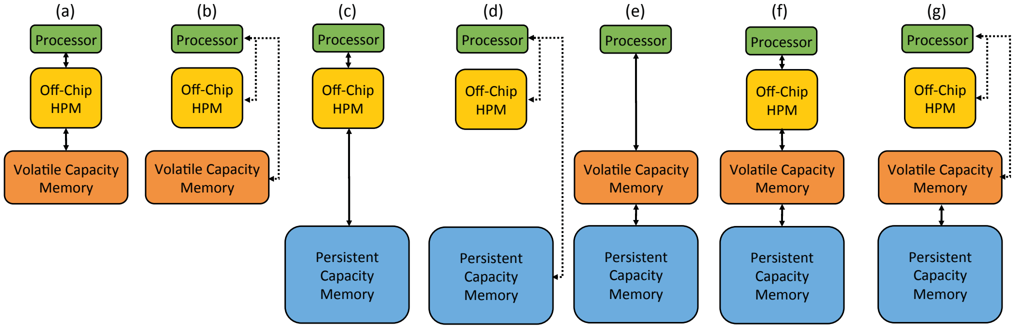


Fig. 2: Selected architectures for emerging memory systems. Solid arrows represent cache-managed memory (i.e., *vertical* organization); dotted arrows represent software-managed memory (i.e., *horizontal* or *flat* organization).

the bandwidth of DDRx. Their performance mainly comes from wide data buses and high data rates. In fact, their access latencies are even slightly higher than that of the conventional DDRx. Also, their high cost restricts their capacity to be orders of magnitude smaller than the other technologies. The second tier of memories includes the mainstream DDRx technology as well as some byte-addressable NVMs with comparable latency, e.g., PCM and STT-MRAM. These memories have moderate performance and a competitive cost per bit. Thus, they determine the capacity of a memory system, and we categorize them as *capacity memory*. One scaling limitation of the volatile capacity memories is their power consumption. Volatile memories need to refresh data periodically, and the required power increases when the capacity increases. Currently, the refresh period of DDR3 is about 7.8 μs [25]. If the next-generation supercomputers only rely on volatile memories for capacity, the total power consumption could be substantial. Recent works have proposed using NVMs to partially replace DDRx because they do not require refresh power and have higher densities. Nevertheless, NVMs suffer from longer access latency and limited endurance. For example, PCM has about 4 and 12 times the read and write latency, respectively, compared to DDR3 [27]. Thus, data management in a heterogeneous memory system becomes critically important for avoiding performance degradation. The last two tiers of memories are *persistent capacity memories*, which have the highest package density but also the lowest performance and may not be byte addressable. The 3D-XPoint and Z-NAND technologies have a much lower latency than that of conventional NAND flash memories. They target to bridge the performance gap between main memory and storage. As the 3D-XPoint can be configured as byte addressable, it could also be an extension to the main memory. In this work, our exploration includes multiple tiers of memory technologies in a heterogeneous memory system.

B. Memory Organization

We mainly explore the *vertical* and *horizontal* organization of multiple memories in a memory subsystem. A *vertical* organization of memories places one memory on the top of another, forming a hierarchy. This organization is similar to the conventional SRAM-cache to DRAM memory. The main purpose of a cache is to keep useful data closer to the processing unit. Therefore, we assume a typical configuration will always place the more performant memory (top tiers in Table I) as a cache to the less performant one. This design direction leads us to those architectural options depicted in solid lines in Figure 2. Different from conventional SRAM caches, these high-performance memories do not provide advantages regarding latency (Table I). Some may even have higher latency than volatile capacity memories. Will this limitation affect the efficiency of a memory system? Also, the capacity of a cache does not contribute to the total memory capacity (Figure 1). What is the cost-effective cache size in a vertical organization for specific workloads? We try to answer these questions from our system exploration.

A *horizontal* organization of memories places one memory next to another, forming a flat memory space. Such memory systems require explicit data management from the software to utilize all memories. For instance, applications may statically place data structures at the initialization phase. During execution, OS kernels or runtimes can also migrate data between memories at various granularity. While the optimal data placement on memories is an NP-complete problem [53], it is possible to explore a smaller mapping space in experiments to reveal design trends. Although a horizontal organization requires additional support from the software, it may only need a fraction of expensive memories compared to a vertical organization. An even stronger motivation for the horizontal organization is that it supports mapping application regions to the appropriate memory kind, which is infeasible in a cache-based organization. What is the sensitivity of a horizontal memory organization to the data mapping? Can a horizontal organization outperform a vertical organization? Our exploration employs a systematic survey of these questions.

III. *Siena* OVERVIEW

We designed *Siena* – a new approach – for fast exploration of these memory system designs. *Siena*, as broached in §I, was motivated by several factors to overcome the limitations of prior approaches (discussed §VI). First, an approach for design exploration must be fast. When considering hundreds or thousands of experiments, we need an approach that can evaluate a design option without significant slowdowns ($> 10,000$x slowdown) as are typical in system simulations. Second, an approach must be flexible in memory system configurations. When considering a wide range of memory system architectures and device parameters, an approach must be more configurable than has been the case over the past two decades when considering only typical cache-based memory hierarchies. Third, an approach must be adaptable to workload characteristics. Because memory system designs may vary dramatically in capacity and performance, the application configurations must similarly be scaled to fit the experiment while maintaining the fidelity of the application workload. Moreover, for software-managed memory options, an approach has to allow exploration of data placement into specific memory spaces. Fourth, an approach must be accurate in revealing the trends of design options, while respecting these other constraints. Clearly, we must balance accuracy with flexibility and speed. However, because we are focusing on the memory system performance, we prioritize the accuracy of the memory system results while optimizing the other components of the approach (e.g., processor model) for speed and flexibility (e.g., easily changing from a GPU model to a CPU model).

Siena uses abstractions in a formal domain specific language (DSL) [43] to represent application behaviors in parameterized models, including memory access, computation, threading, and dependency. It takes abstract machine models to configure a general memory system, uses memory simulators to model different memory technologies, and orchestrates the interplay among memories. *Siena* bridges the application and the memory subsystem with an integrated framework that can explore memory-level parallelism and memory-computation overlapping while honoring the intrinsic dependencies in application memory accesses.

A. *Application Abstraction*

An application abstraction is similar to a "minimal" description of the original application, consisting of parameters, data structures, kernels, and control flows. Each kernel contains quantitative information of memory accesses, computation, and communication. These kernels are connected through a control flow that depends on the input problem during "execution". Abstractions also use optional traits to capture additional characteristics of workload behaviors, including memory accesses patterns, data layout, and data dependencies. Application abstractions are created from automatic compiler analysis while the optional traits are manually added based on runtime profiling results. We report the details of model creation, workload generation, and validation in [36]. We utilize the application abstraction in *Siena* to generate configurable memory workloads for the system exploration.

The application abstraction models memory workload behaviors by providing a data-centric view of each kernel. A kernel has multiple data objects that describe the behavior of a particular phase through their properties and interactions. The model uses specific constructs and memory traits in

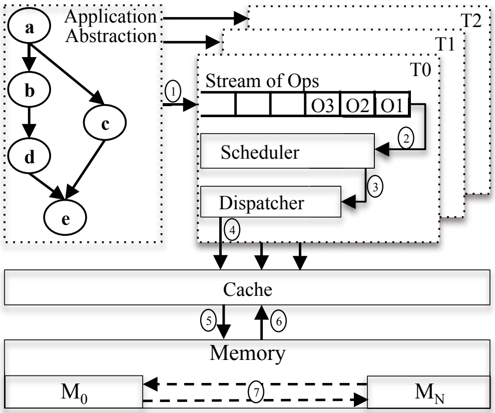


Fig. 3: The life cycle of memory traffic in *Siena* framework: originated from application models (①), dispatched out-of-order (②③④), and serviced by memory subsystem (⑤⑥⑦).

the DSL to capture the properties of a data object, such as data layout, access patterns, and access types, as well as interactions between data objects, such as computation and data dependency. A kernel can access different data structures in different patterns by traversing from one data object to another. As these data objects are used to separate memory access behaviors, they can access the same allocation in the memory. The model uses separate constructs in the DSL to describe these memory allocations and keeps them orthogonal from the workload behavior in the kernels. These memory allocations can be explicitly managed by *Siena* to be placed on a particular memory in the memory subsystem. The model also captures thread interleaving in data accesses. Multi-threaded accesses to the memory are critical for exploiting high-memory bandwidth. However, it may increase row-buffer conflicts or cause contention on the memory resource. Thus, *Siena* also models the back pressure from the memory system in multi-threaded workloads. *Siena* reconstructs application behaviors from this abstraction into streams of pseudo operations to drive simulations of memory subsystems.

B. *Memory Subsystem*

Siena facilitates flexible configurations of a general heterogeneous memory system. The architecture of a memory system is specified in an abstract machine model in the same DSL as the application abstraction. The memory configuration describes the organization of different memories, i.e., vertical or horizontal, as well as their parameterized capacity, associativity, access latency, bandwidth, etc. *Siena* takes a machine model and constructs the specified memory system. In particular, *Siena* supports using proper memory simulators ([23], [41], [45]), for the particular memory technologies in use, to model the behavior of each component in the memory subsystem. *Siena* also manages the data mapping onto different memories and the data migration among these memories.

For a vertical organization of memory M_i and M_j, we model M_i as an associative cache to M_j. When the requested data is not available in M_i, *Siena* evicts a cache line (of configurable size in the machine model) and sends memory requests to M_j. *Siena* maintains a buffer of pending requests so that multiple concurrent requests can be consolidated into fewer memory transactions. *Siena* also automatically handles the difference in access granularity when migrating data between two memories. For a horizontal organization, *Siena* manages flexible data mapping schemes by reading in a mapping model in the same DSL as the application abstraction. At the initialization phase of the "execution", *Siena* places data structures according to the specified mapping scheme while ensuring the data placement is feasible under the capacity constraint of each memory. During the "execution", *Siena* maintains the most current data mapping in a mapping table. *Siena* performs a lookup in the table to identify the target memory for sending a memory request.

C. Memory Interplay

Siena models the interplay between memories through the shared hardware resources and workload behaviors. Typical hardware resources include those buffers that hold outstanding cache misses and instructions [28]. For instance, if a buffer is occupied with outstanding requests to a slower memory, even independent requests to a faster memory can be delayed. Workload behaviors impact the memory interaction through data dependencies such that accesses to data structures in memory M_i may need to wait for data fetched from M_j. If M_j has lower performance than M_i, it could also delay memory accesses to M_i, lowering the overall performance. From this perspective, *Siena* models a simple core pipeline similar to [23] and also models the contention on shared resources and the constraints of data dependencies. We note that although these effects impact the application performance on heterogeneous memory systems, conventional trace-based simulations cannot reflect these dynamic interactions, which may neglect the coupling effect among memories and result in higher errors as the number of cores increases on-chip [44].

To model overlapping between data transfer and computation, *Siena* configures a reorder buffer of depth D_p to control the parallelism among operations. During a simulation, *Siena* checks each operation in the buffer to determine whether its required hardware resources and dependent data are both available. The hardware resources include functional units, e.g., arithmetic or load/store ports, as well as the buffers that hold outstanding cache misses. For simplicity, *Siena* only models three types of operations, i.e., computation, load, and store, and the throughput of each kind of operation is configured through the machine model. *Siena* honors the data dependency as specified in the application abstraction. A data object in a kernel could have dependencies on a list of other data objects. *Siena* delays dispatching an operation and keeps it in the buffer until all its dependencies have been satisfied. Upon the response of a memory request, *Siena* updates the resolved dependency for pending operations. *Siena* also orchestrates the memory traffic to the memory subsystem. For each memory request, e.g., a load miss or a dirty write back, *Siena* performs a lookup in the mapping table to determine in which memory the data currently resides. Then, it directs memory requests to the mapped memory and also updates the buffers of outstanding cache misses. Finally, *Siena* handles contention on the memory system from multiple threads. When too many requests are in-flight, the memory system could reject some requests, causing a thread to progress slower than others at runtime.

We illustrate the workflow of *Siena* through the life cycle of a memory request in Figure 3. From an application abstraction (①), *Siena* generates a stream of pseudo operations ($MemOp$ or $CompOp$) for each thread (T1,T2,T3). These operations (O1,O2,O3) wait in the buffer until *Siena* dispatches them out-of-order (②,③,④). Operations are removed from the buffer when their transactions are completed in-order. If an operation is of type $MemOp$, it needs to go through the cache hierarchy (⑤), which is configurable through the machine model and might be implemented by a HPM. When an operation results in memory transactions, *Siena* orchestrates memory requests to the memory subsystem, handling data mapping, request consolidation, and data migration accordingly.

IV. Evaluation Methodology

For the experiments, we configure a test system that consists of 32 cores. Each core has a clock speed of 2 GHz and can issue eight pseudo operations per cycle. At each cycle, each core can dispatch at most four computation, two load and store operations. All experiments in this work use a 128-wide reorder buffer and follow an in-order issue, out-of-order dispatch, and in-order retirement pipeline. Each core has a private 1KB cache. We model hits in the first-level cache as four CPU cycles. All cores share the last-level cache (LLC) that is backed by a high-performance memory. The L1 cache uses LRU replacement policy, write-back, write-allocate and 8-way associativity. LLC uses the same policies but with 16-way associativity. We varied the capacity of LLC for the experiments of vertical organizations. We also modeled a fill buffer for L1 caches and configured its capacity to hold at most 10 concurrent misses. This configuration is mostly based on the Intel Haswell microarchitecture [8] with simplified the cache and core modelling to speed up the exploration. All the experiments use the same core-side configuration, and we focus on comparing relative changes among different memory subsystems.

We create application models in two steps. First, a skeleton model is created from the source code by a research compiler that performs static analysis to extract quantitative information of computation, memory access, and communication [29]. Then, we manually extend the model with more specific memory traits that include access patterns and data dependency based on runtime profiling results and application knowledge [34], [36].

To model HPMs and volatile capacity memories, we use the Ramulator [23] memory simulator. We assume persistent memories in DIMM form factor and use NVDIMMSim [45] simulator. We configure the granularity of data migration between volatile memories to be 64 bytes and to/from persistent memories to be 4096 bytes. For volatile memories, we use row-interleaved, FR-FCFS scheduling, and open page policy. We model DDR4 as one-channel single-rank memory chips in an 8gbx16 organization with a 64-bit data bus, $2,400$ MT/s data rate, and 16–16–16 timing and HBM as eight-channel single-rank memory chips in 1gb, 2gb, and 4gb organizations with a 128-bit data bus, $1,000$ MT/s data rate, and 7–7–7 timing [23]. For persistent memory, we use multiple packages of 4 GB density. We advance the memory simulators and the core pipeline respectively to their clock rate.

We summarize the list of applications ([1], [16], [31]) in experiments in Table II. The last column reports the parameterized problem size in application abstraction. Each experiment includes one additional iteration to warm up the system for measurement. We only report the measurement obtained from actual execution. If not specified otherwise, we run experiments with 32 threads. Our results use the total simulated CPU cycles for comparing performance.

TABLE II: List of applications.

Application	Description	Parameters of Application Size
spmul	Sparse linear algebra	$12nnz$
stream	Memory bandwidth	$24n$
miniFE	Finite-element solver	$\prod_{i=1}^{3}(d_i+1)(8nnz_{row}+56)$
jacobi3d	Iterative solver	$16n^3$
npb.mg	Multi-grid smoother	$8\sum_{i=1}^{logN}(2^i+2)^3$
conv	Convolution filter	$8(n_1^2+n_2^2)$
laplace	five-point stencil	$16n^2$

V. Experiments

We present the results of system exploration in this section. Initially, a memory system must meet the minimum requirement on capacity as indicated by all applications (Table II). For a fixed memory capacity, a system design could optimize for multiple objectives such as performance and cost. Systematically, we evaluate the trade-off between the HPM ratio and the performance changes under vertical and horizontal organizations of memories. Our results show that a vertical organization can improve performance up to that of the HPM in the system. In contrast, a horizontal organization can use less HPM and can reach even higher performance than that of the HPM when the distribution of memory traffic balances the performance gap between memories. Also, attention must be paid to avoid the coupling effects among memories, which can negate the advantage of HPMs.

A. Vertical Organization

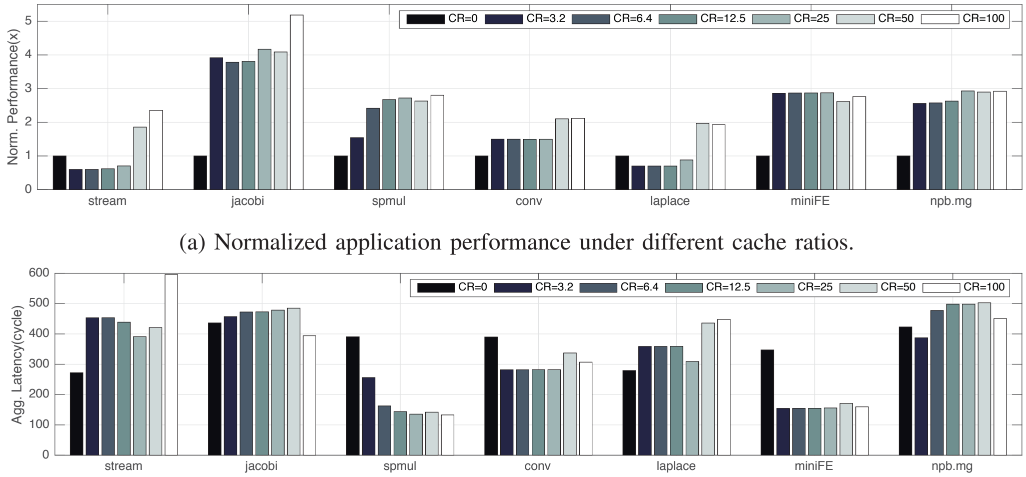
Our first set of experiments explore the impact of HPM ratio in vertical organizations of heterogeneous memory systems. We construct seven memory subsystems. Each system consists of a volatile capacity memory that simulates a single-channel DDR4 memory. Six out of the seven systems also have a HPM that simulates a HBM. The ratio of HPM in these six systems ranges from 3.2% to 100% at a doubling rate. We use the system that has no HPM as the baseline. We refer to these systems as CRi, where $i = 0, 3.2...100$. On these systems, we run a set of seven applications. Each application uses a large input problem. We normalize the performance of these systems to that on the baseline system ($CR0$) and report the results in Figure 4.

Our experimental results show that application performance in vertical organizations reaches up to that on a full HPM system. It is clear from Figure 4a that all applications achieve the highest performance on the system with the largest HPM ratio. Note that there exist minor dips from $CR50$ to $CR100$, which could be a result of using different chip organization for implementation. As a full HPM system is unlikely an affordable option, system designers first need to identify feasible options, e.g., from Figure 1 and Table II, and then choose options that balance the increased cost of HPM and the improvement in applications.

Applications in the experiments demonstrate three levels of sensitivity to the change of HPM ratios. First, not all applications can immediately benefit from using HPM. Some applications, such as stream and laplace, only start showing performance improvement when the HPM ratio reaches a threshold value. In fact, they exhibit lower performance when the ratio is small. This performance degradation is due to the low data reuse in HPM caches. Consequently, there is little performance benefit to trade-off the overhead of managing a cache. Second, some applications can benefit from a small HPM cache and are insensitive to the increase of HPM ratios. For instance, convolution, miniFE, and MG show approximately 1.5, 3, and 2.8 times improvement in performance, respectively, at the smallest HPM ratio. Beyond that, their performance remains mostly stable on systems with larger ratios. This insensitivity is favorable for vertical organizations because only a small portion of HPM is needed for applications to benefit from the HPM. We note that convolution has a relatively lower performance improvement than the others because of its higher compute intensity, while miniFE is more sensitive to memory performance. The third group of applications is most sensitive to HPM ratios, showing increasing performance when the ratio increases. System options for such applications can be narrowed down to those options that meet both a target performance and a target capacity.

Our second set of experiments focuses on memory-level parallelism (MLP) that applications can benefit from HPMs. HPM-based caches are different from the conventional SRAM-based caches because, typically, SRAM are advantageous in regards to both latency and bandwidth compared to DRAM. In contrast, HPMs (Table I) are only advantageous in bandwidth, compared to those volatile capacity memories. Therefore, selecting appropriate memory designs requires understanding the sensitivity of applications to latency and MLP.

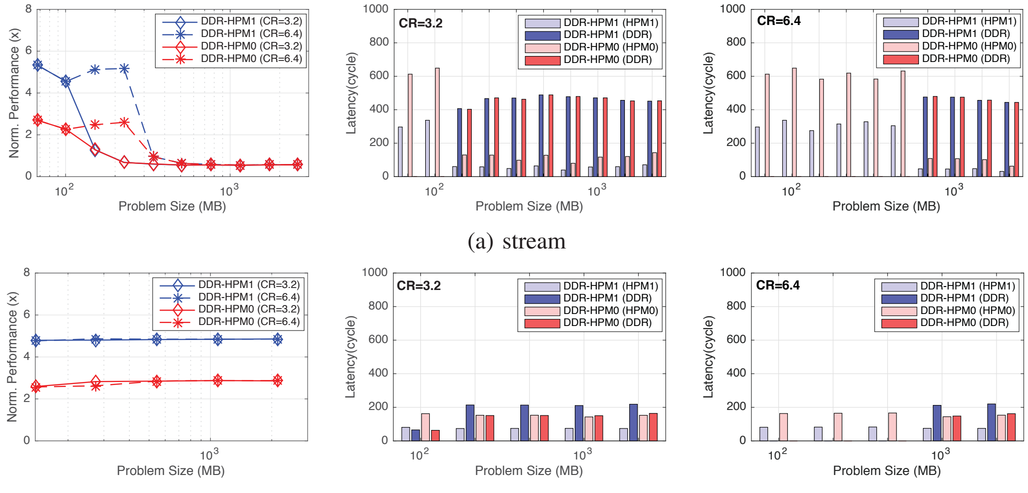
We calculate the aggregated latency as $(N_{M1} * Lat_{M1} + N_{M0} * Lat_{M0})/(N_{M1} + N_{M0})$, where $M1$ is a HPM cache to $M0$, N_{M_i} is the number of reads, and Lat_{M_i} is the measured average latency to memory i. This latency estimates



(a) Normalized application performance under different cache ratios.

(b) Aggregated read latency under different cache ratios.

Fig. 4: Comparison of performance and latency under different cache ratios in vertical organizations.



(a) stream

(b) miniFE

Fig. 5: Comparison of performance and latency under different workloads and cache size in vertical organizations.

the average time that a load will take to complete if it misses in the previous cache level. We report the aggregated latency in Figure 4b. We deduce the impact of MLP in each application by comparing changes in the performance and the aggregated latency. MLP indicates the number of memory requests that can be concurrently served. In general, the more memory requests in-flight to saturate the memory bandwidth, the better an application can exploit a high-bandwidth memory. Also, when the number of in-flight requests increases, the latency of completing a single request can also increase. Thus, if both latency and performance increase (by comparing Figure 4a and 4b), we are confident in attributing the improvement to better MLP. For instance, jacobi shows both increased performance and latency on system $CR3.2$, $CR6.4$, and $CR12.5$, indicating that its benefit from improved MLP exceeds the increased latency. On the contrary, spmul and miniFE both exhibit performance improvement that is mostly proportional to the reduced latency on the test systems. Therefore, they may not exploit as much MLP as jacobi in these experiments.

To further evaluate the impact of HPM ratios on MLP in applications, we configure six memory systems in the second set of experiments. Three systems use the same memory options as in the first set of experiments. We refer to them as DDR-HPM0 (CR_i). The additional three systems use an even higher performance memory that has a double bandwidth of HBM, and we refer to them as DDR-HPM1 (CR_i). On each system, we scale up the problem size for each application. We normalize their performance to that on their respective baseline system DDR-HPM$_i$ ($CR0$). We report the performance and the average access latency to each memory in Figure 5.

We use the change of MLP to categorize whether an application can efficiently utilize HPMs in vertical system options. When a decreased performance accompanies a reduced latency to HPMs, we infer a decreased MLP in HPMs. From the experimental results, stream and miniFE represent two types of applications whose MLP changes differently when their problem sizes increase relative to HPM ratios. Stream only benefits from HPM when the problem size is smaller than the HPM capacity. The average latencies to HPM0 and HPM1 (translucent blue and red bars in the middle and right panels of Figure 5a) significantly decrease at the problem size when the overall performance starts decreasing (the left panel of Figure 5a). We conclude that vertical organizations of memories are likely to have one type of memory underutilized, i.e., either HPMs or volatile capacity memories, depending on the relative problem size and cache size for such applications.

On the other hand, the utilization of HPMs remains stable regardless of the change of problem size and HPM ratio in applications like miniFE. We do not observe any abrupt performance degradation from the left panel of Figure 5b, even when its memory footprint is larger than the HPM capacity. The changes in the latency to HPMs are also distinctively differ from stream. We observe that latencies to HPMs (translucent blue and red bars in the middle and right panels of Figure 5b) remain nearly constant when the workload changes, implying a relatively stable utilization of HPMs. We note that an ideal use case should have improved utilization of HPMs when the workloads scale up. However, we do not observe such results from our exploration.

B. Horizontal Organization

A horizontal organization places different memories side by side to form a flat memory space. Such systems rely on the software support to manage data mapping across multiple memories. In this section, we explore (1) the system sensitivity to data mapping and (2) the optimal distribution of traffic under different memory options.

1) Data Mapping: To explore data mapping in horizontal organizations, we first construct six memory systems. The first two systems represent the proposed architecture depicted in Figure 2b, which consist of a HPM and a volatile capacity memory, denoted as DDR-HPM0 and DDR-HPM1, respectively. The third and fourth systems are similar to the first two but replace the volatile capacity memory with a non-volatile capacity memory that has approximately $1,000$ times higher latency. They represent the architecture depicted in Figure 2d. We refer to these two systems as NVM-HPM0 and NVM-HPM1. We also construct two baseline systems, where the first one only consists of the volatile capacity memory, while the second one only consists of the non-volatile capacity memory. We list the specification of each system in the second column of Table III.

TABLE III: Sweep data mapping on horizontal system options

			Data Structure				
architecture	mem. spec.	mapping	$colind$	$value$	$rowptr$	x	y
	DDR	m1	DDR	DDR	DDR	DDR	DDR
Fig.2b	DDR-HPM0	m2	DDR	DDR	HPM0	DDR	HPM0
		m3	DDR	DDR	DDR	HPM0	DDR
		m4	DDR	DDR	HPM0	HPM0	HPM0
Fig.2b	DDR-HPM1	m5	DDR	DDR	HPM1	DDR	HPM1
		m6	DDR	DDR	DDR	HPM1	DDR
		m7	DDR	DDR	HPM1	HPM1	HPM1
Fig.2d	NVM-HPM0	m8	NVM	NVM	HPM0	NVM	HPM0
		m9	NVM	NVM	NVM	HPM0	NVM
		m10	NVM	NVM	HPM0	HPM0	HPM0
Fig.2d	NVM-HPM1	m11	NVM	NVM	HPM1	NVM	HPM1
		m12	NVM	NVM	NVM	HPM1	NVM
		m13	NVM	NVM	HPM1	HPM1	HPM1
	NVM	m14	NVM	NVM	NVM	NVM	NVM

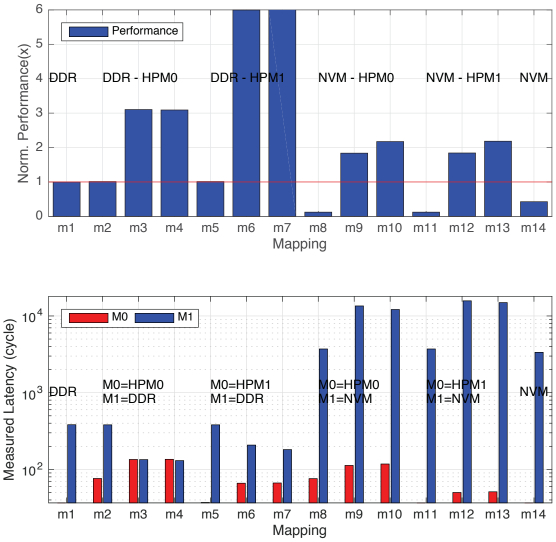


Fig. 6: Sweep data mapping in horizontal organizations. The top panel presents the performance normalized to the baseline $m1$. The bottom panel presents the average measured latency to memory M0 and M1, respectively.

Experiments on data mapping can result in a large search space that grows exponentially when the number of data structures and memories increases. In the first set of experiments, we use a linear algebra benchmark that performs matrix-vector multiplication on a sparse matrix consisting of 2M rows and 182M non-zeros. Sparse matrix-vector multiplication is a common operation in numerical solvers like conjugate gradient solvers in scientific applications. It is often a performance bottleneck due to its high-memory intensity and irregular access pattern. The benchmark consists of five main data structures (Table III) that exhibit a mixture of different data sizes, access patterns, and data dependency. On the four horizontal systems, there are a total of 128 possible mappings. We decided that the largest data structures ($values$ and $colind$) should always be mapped to the most cost-effective memory to reduce HPM cost. Due to limited space, we only show a subset of mappings on each system option in Table III. We denote each mapping with mi in the third column and compare their performance in Figure 6.

The constructed systems exhibit different sensitivity to data mapping. We identify three categories of mapping based on their relative performance to the baseline $m1$. The first type of mapping includes $m3$, $m4$, $m6$, $m7$, $m9$, $m10$, $m12$, and $m13$. They all achieve a higher performance than the baseline

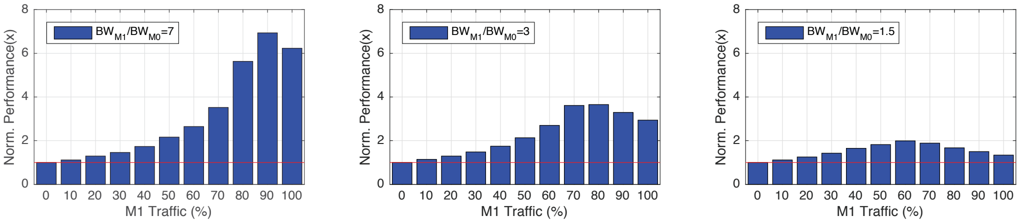


Fig. 7: Sweep traffic distribution in three memory systems. Each system consists of a capacity memory M0 and a high-performance memory M1. Bars present the performance normalized to 0% traffic to M1.

(indicated by the red line in Figure 6). Among them, $m3$ and $m4$ show a similar performance, but $m3$ achieves this by only placing x in HPM0, while $m4$ places all x, y,and $rowptr$ in HPM0. Using $m3$ only requires one-third the HPM capacity that is required by $m4$. By comparing $m3$ and $m6$, we observe a twofold improvement in performance. $m3$ and $m6$ place the same data structure on HPM0 and HPM1, respectively, which indicates that it is only sufficient to speed up accesses to x in a horizontal organization. Interesting, we do not observe such significant improvement in performance when HPMs are placed next to NVMs in systems NVM-HPM0 and NVM-HPM1. The performances of $m9$ and $m12$ in Figure 6 are very similar, unlike $m3$ and $m6$. The different sensitivities of NVM and DDR, when both are placed horizontally next to HPM, indicate that significantly slower memories could have *back-pressure* to fast memories in the system, hiding their advantages. In fact, the performance difference between $m9$ and $m10$ proves that NVM is throttling the performance. Recall that $m4$ does not show improvement compared to $m3$, but now on the NVM-based system, this mapping shows improvement in performance by comparing $m9$ and $m10$. When memories constitute a memory subsystem, they interfere with each other through shared buffers such that requests waiting for responses from NVM will occupy the buffer longer than requests to DDR or HPM. As the gap between DDR and HPM is not as dramatic as the gap between HPM and NVM, as shown in the bottom panel of Figure 6, the back-pressure from NVM has a more obvious throttling effect on the performance of HPM. Besides, x in fast memory has a data dependency on $colind$ in slow memory. This application characteristic further tightens the coupling of these memories.

The performance of the second category of mappings is similar to that of the baseline system, namely, $m2$ and $m5$. They place $rowptr$ and y in the high-performance memory and the remaining data structures onto the capacity memory. These mappings do not result in significant changes in either the number of memory transactions or the average access latency. Thus, we observe a performance similar to the baseline, indicating that such mappings are not effective in exploiting HPM. Finally, the third category of mappings includes $m8$, $m11$, and $m14$. They all show a lower performance than that of the baseline. The bottom panel of Figure 6 shows that $m8$, $m11$, and $m14$ all have a latency similar to NVM (blue bars), which is approximately $1,000$ time that of other memories. The performance of $m14$ is within what is expected as it places all data structures onto the lower-performance memory. However, $m8$ and $m11$ are surprisingly slower than $m14$. The statistics from simulators show that $m8$ and $m11$ result in a higher number of transactions. Note that NVM has block access granularity in the setup, and concurrent reads to the same page are consolidated. This increase in transactions is likely a result of the changes in data mapping, which reduces the probability of transaction consolidation.

2) Memory Traffic: Our second exploration of horizontal organization investigates the impact of memory traffic distribution on system performance. We construct three heterogeneous memory subsystems, each consisting of a capacity volatile memory (M0) and an HPM (M1). M0 simulates DDR4 with one, two, and four channels in the three systems. M1 simulates a high-bandwidth memory. In these systems, the bandwidth of M1 is 7, 3, and 1.5 times that of M0, respectively. We use a synthetic benchmark to control the distribution of memory traffic to each memory. We sweep the distribution from 0% to 100% traffic to M1 at the step of 10%. We report the performance normalized to that of 0% distribution on each system in Figure 7.

Our experimental results show that each test system can achieve higher performance than the peak performance of HPM at certain traffic distributions. For instance, the middle panel of Figure 7 shows that when 70% traffic goes to HPM, the system can achieve nearly 3.8 times speedup compared to the baseline (0%). The peak performance of HPM, however, is only three times that of the baseline. The optimal distribution of traffic among the two memories depends on their relative bandwidth. When there is a considerable gap in their bandwidth, it is more difficult to reach high performance. For instance, the left panel of Figure 7 shows that there is only one distribution of traffic in the experiment, i.e., 90%, that has reached higher performance than the full HPM traffic (100%). In contrast, the middle panel has three distributions, i.e., 70%, 80%, and 90%, that have reached a higher performance than the full HPM traffic. Moreover, the right panel of Figure 7 has seven qualified distributions. The different numbers of qualified traffic distribution on these systems demonstrate that when the gap between two memories gets smaller, there is a higher probability of reaching a higher performance than that of HPM. Our exploration results show that horizontal organizations not only have advantages in aggregated capacity but, more importantly, can also achieve higher performance than vertical organizations whose performance is limited by that of HPMs.

C. Discussion and Insights

We note that our exploration is based on the assumption that the core-side architecture, e.g., network-on-chip (NoC) bandwidth, should have been sufficiently optimized to avoid throttling HPMs [49]. Under this assumption, we argue that vertical organizations would be a preferred memory option when workloads can sustain their utilization of HPMs on typical problem sizes. In this work, we use MLP as a metric for the HPM utilization. Horizontal organizations can potentially achieve higher performance than vertical organizations. However, we have also demonstrated the high complexity of data management that is necessary from software support.

We recommend using the sensitivity to HPM ratios and the HPM utilization to select memory organizations. We summarize the configuration for each application based on the exploration results in Table IV. We choose vertical organizations when the workloads are insensitive to HPM ratios; i.e., they only require a fixed small HPM to improve performance. We select horizontal organizations and only use the capacity memory but not HPM when the workloads need a large HPM ratio in vertical organizations and show low HPM utilization on smaller ratios. Finally, applications whose performance increases when the HPM ratio increases in vertical organizations may also benefit from horizontal organizations. We recommend choosing memories whose performance gap is no larger than the traffic distribution to different memories to improve overall performance and to avoid back pressure from slow memory in horizontal organizations.

TABLE IV: Qualitative summary of results for applications.

Application	HPM utilization	HPM Ratio	Memory Organization
npb.mg	high	insensitive	vertical
jacobi3d	high	scaling	vertical
conv	medium	insensitive	vertical
miniFE	medium	insensitive	vertical
spmul	medium	scaling	horizontal, vertical
stream	low	sensitive	horizontal
laplace2d	low	sensitive	horizontal

Furthermore, a complex memory subsystem could be a hybrid of vertical and horizontal organizations (Figure 2g). Such systems would require software support not only for data mapping but also for kernel mapping. Kernels in one application may prefer different memory organizations. Changing from a vertical organization requires data to be synchronized in all memories, which should be supported by software. Thus, runtime support is indispensable for the successful adoption of complex memory systems.

VI. Related Work

We categorize related works on system design exploration into analytical modelling, emulation, and simulation.

Analytical models for design space exploration are generally fast in pruning promising design trends [9], [25]. Knyaginin et al. [25] used the cache reuse profiles of workloads to predict memory traffic to each memory level and then studied the design dimensions in resource partition, allocation, and data placement. Their work assumed static row buffer hit rates and only focused on single-threaded workload. Different from their work, our approach uses memory simulations to model row buffer hit rates dynamically and considers contention and back-pressure on the memory system in those multi-threaded workloads.

Emulations run on native hardware and thus are still fast for exploring new technologies [4], [11], [35], [46]. Hardware emulators use accelerators, such as FPGA, to execute applications at high speed. Nevertheless, their exploration space is limited to the resource of the accelerators in use. Software emulators do not rely on hardware resource but may introduce overhead that is comparable or even higher than certain high-performance memories. Thus, they are often used to model slow memories. Doudali et al. [11] used CPU throttling to reduce the bandwidth and increase latency to one NUMA domain. Such emulation may be infeasible for those memories with asymmetric latency or for different memory organizations.

Simulation is the most important approach for designing new hardware [2], [3], [22], [30], [38], [52]. Cycle-accurate simulators can achieve higher accuracy than analytical models and emulations. Nevertheless, detailed simulations often have a speed of hundreds of kilo-instructions per second (KIPS). Thus, using full-system simulations to sweep a broad design space is prohibitively expensive. Recent works have proposed simulators with different areas of interest such as volatile memories [23], [41], non-volatile memories [37], circuit-level [10], memory controllers [15], memory organization [12], and even the entire memory hierarchy [45]. Our approach takes advantage of this extensive range of simulators. We provide a framework that systematically configures a heterogeneous memory system, selects appropriate simulators for the memory options of interest, and orchestrates the interplay among them.

VII. Conclusions

The design space for heterogeneous memory systems is a complex, multidimensional problem. In this work, we systematically explored the organization of heterogeneous memory systems on *Siena* framework. Based on the results, we recommend selecting vertical organizations for applications that are insensitive to HPM ratios and can sustain HPM utilization at different workloads. We also show that horizontal organizations couple memories through shared hardware resources and application characteristics and thus should balance memories to eliminate the back pressure from the slower memories, which can negate the advantage of faster memories.

Acknowledgment

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

References

[1] Rodinia:Accelerating Compute-Intensive Applications with Accelerators. https://www.cs.virginia.edu/∼skadron/wiki/rodinia/index.php/, 2017. [Online; accessed 15-Janurary-2018].

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[3] Trevor E Carlson, Wim Heirmant, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[4] Adrian M Caulfield, Joel Coburn, Todor Mollov, Arup De, Ameen Akel, Jiahua He, Arun Jagatheesan, Rajesh K Gupta, Allan Snavely, and Steven Swanson. Understanding the impact of emerging non-volatile memories on high-performance, I/O-intensive computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[5] W. Cheong, C. Yoon, S. Woo, K. Han, D. Kim, C. Lee, Y. Choi, S. Kim, D. Kang, G. Yu, J. Kim, J. Park, K. W. Song, K. T. Park, S. Cho, H. Oh, D. D. G. Lee, J. H. Choi, and J. Jeong. A flash memory controller for 15us ultra-low-latency SSD using high-speed 3D NAND flash with 3us read time. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 338–340, 2018.

[6] Jin Hee Cho, Jihwan Kim, Woo Young Lee, Dong Uk Lee, Tae Kyun Kim, Heat Bit Park, Chunseok Jeong, Myeong-Jae Park, Seung Geun Baek, Seokwoo Choi, et al. A 1.2 V 64Gb 341GB/S HBM2 stacked DRAM with spiral point-to-point TSV structure and improved bank group data control. In *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pages 208–210. IEEE, 2018.

[7] HMC Consortium. Hybrid Memory Cube Specification 2.1. http://hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf, 2015. [Online; accessed 22-May-2018].

[8] Intel Coorporation. Intel 64 and IA-32 architectures optimization reference manual, 2016.

[9] Kenneth Czechowski and Richard Vuduc. A theoretical framework for algorithm-architecture co-design. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 791–802. IEEE, 2013.

[10] X. Dong, N. P. Jouppi, and Y. Xie. A circuit-architecture co-optimization framework for exploring nonvolatile memory hierarchies. *ACM Transactions on Architecture and Code Optimization*, 10(4):1–22, 2013.

[11] Thaleia Dimitra Doudali and Ada Gavrilovska. Comerge: toward efficient data placement in shared heterogeneous memory systems. In *Proceedings of the International Symposium on Memory Systems*, pages 251–261. ACM, 2017.

[12] Bharan Giridhar, Michael Cieslak, Deepankar Duggal, Ronald Dreslinski, Hsing Min Chen, Robert Patti, Betina Hold, Chaitali Chakrabarti, Trevor Mudge, and David Blaauw. Exploring DRAM organizations for energy-efficient and resilient exascale memories. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 23. ACM, 2013.

[13] R. Hadidi, B. Asgari, B. A. Mudassar, S. Mukhopadhyay, S. Yalamanchili, and H. Kim. Demystifying the characteristics of 3D-stacked memories: A case study for Hybrid Memory Cube. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–75, Oct 2017.

[14] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3D XPoint technology. *Proceedings of the IEEE*, 105(9):1822–1833, 2017.

[15] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wenisch, and Aniruddha N Udipi. Simulating DRAM controllers for future system architecture exploration. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 201–210. IEEE, 2014.

[16] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[17] Huynh Phung Huynh, Andrei Hagiescu, Weng-Fai Wong, and Rick Siow Mong Goh. Scalable framework for mapping streaming applications onto multi-GPU systems. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12)*, pages 1–10, 2012.

[18] JEDEC JESD229-2. Wide I/O 2 (WideIO2), 2014. [Online; accessed 22-May-2018].

[19] JEDEC JESD235A. High bandwidth memory (HBM) DRAM. *JEDEC Solid State Technology Association*, Nov 2015.

[20] JEDEC JESD250. Graphics double data rate 6 (GDDR6) SGRAM standard. *JEDEC Solid State Technology Association*, Jul 2017.

[21] JEDEC JESD79-4B. DDR4 SDRAM standard. *JEDEC Solid State Technology Association*, Jun 2017.

[22] M. Jung, I. Ellis H. Wilson, W. Choi, J. Shalf, H. M. Aktulga, C. Yang, E. Saule, U. V. Catalyurek, and M. Kandemir. Exploring the future of out-of-core computing with compute-local non-volatile memory. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Denver, Colorado, 2013. ACM.

[23] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.

[24] Young-Ju Kim, Hye-Jung Kwon, Su-Yeon Doo, Yoon-Joo Eom, Young-Sik Kim, Min-Su Ahn, Yong-Hun Kim, Sang-Hoon Jung, Sung-Geun Do, Chang-Yong Lee, et al. A 16Gb 18Gb/S/pin GDDR6 DRAM with per-bit trainable single-ended DFE and PLL-less clocking. In *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pages 204–206. IEEE, 2018.

[25] Dmitry Knyaginin and Per Stenstrom. Rock: a framework for pruning the design space of hybrid main memory systems. In *Proceedings of the International Symposium on Memory Systems*, pages 337–347. ACM, 2017.

[26] M. H. Kryder and K. Chang Soo. After hard drives: What comes next? *IEEE Transactions on Magnetics*, 45(10):3406–3413, 2009.

[27] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 2–13. ACM, 2009.

[28] Chang Joo Lee, Veynu Narasiman, Onur Mutlu, and Yale N Patt. Improving memory bank-level parallelism in the presence of prefetching. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 327–336. ACM, 2009.

[29] Seyong Lee, Jeremy S Meredith, and Jeffrey S Vetter. Compass: A framework for automated performance modeling and prediction. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pages 405–414. ACM, 2015.

[30] G. H. Loh. 3D-stacked memory architectures for multi-core processors. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 453–464, 2008.

[31] NASA. NAS parallel benchmarks. http://www.nas.nasa.gov/Resources/Software/npb.html.

[32] Young Paik. Developing extremely low-latency NVMe SSDs. *Flash Memory Summit*, Aug 2017.

[33] Ki-Tae Park, Sangwan Nam, Daehan Kim, Pansuk Kwak, Doosub Lee, Yoon-He Choi, Myung-Hoon Choi, Dong-Hun Kwak, Doo-Hyun Kim, Min-Su Kim, et al. Three-dimensional 128 Gb MLC vertical NAND flash memory with 24-WL stacked layers and 50 MB/s high-speed programming. *IEEE Journal of Solid-State Circuits*, 50(1):204–213, 2015.

[34] 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 *Proceedings of the 2017 ACM SIGPLAN International Symposium on Memory Management*, pages 82–91. ACM, 2017.

[35] Ivy Bo Peng, Stefano Markidis, Erwin Laure, Gokcen Kestor, and Roberto Gioiosa. Exploring application performance on emerging hybrid-memory supercomputers. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 473–480. IEEE, 2016.

[36] Ivy Bo Peng, Jeffrey S. Vetter, Shirley V. Moore, and Seyong Lee. Tuyere: Enabling scalable memory workloads for system exploration. In *Proceedings of the 27th International Symposium on High-Performance*

Parallel and Distributed Computing, HPDC '18, pages 180–191, New York, NY, USA, 2018. ACM.

[37] Matt Poremba and Yuan Xie. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pages 392–397. IEEE, 2012.

[38] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable high performance main memory system using phase-change memory technology. *ACM SIGARCH Computer Architecture News*, 37(3), 2009.

[39] Kwangmyoung Rho, Kenji Tsuchida, Dongkeun Kim, Yutaka Shirai, Jihyae Bae, Tsuneo Inaba, Hiromi Noro, Hyunin Moon, Sungwoong Chung, Kazumasa Sunouchi, et al. 23.5 a 4gb lpddr2 stt-mram with compact 9f2 1t1mtj cell and hierarchical bitline architecture. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 396–397. IEEE, 2017.

[40] ND Rizzo, D Houssameddine, J Janesky, R Whig, FB Mancoff, ML Schneider, M DeHerrera, JJ Sun, K Nagel, S Deshpande, et al. A fully functional 64 mb DDR3 ST-MRAM built on 90 nm CMOS technology. *IEEE Transactions on Magnetics*, 49(7):4441–4446, 2013.

[41] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.

[42] Amit Sabne, Putt Sakdhnagool, and Rudolf Eigenmann. Scaling large-data computations on multi-GPU accelerators. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS'13)*, pages 443–454, 2013.

[43] Kyle L Spafford and Jeffrey S Vetter. Aspen: a domain specific language for performance modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 84. IEEE Computer Society Press, 2012.

[44] Sadagopan Srinivasan, Li Zhao, Brinda Ganesh, Bruce Jacob, Mike Espig, and Ravi Iyer. CMP memory modeling: How much does accuracy matter? 2009.

[45] Jim Stevens, Paul Tschirhart, Mu-Tien Chang, Ishwar Bhati, Peter Enns, James Greensky, Zeshan Chisti, Shih-Lien Lu, and Bruce Jacob. An integrated simulation infrastructure for the entire memory hierarchy: Cache, dram, nonvolatile memory, and disk. *Intel Technology Journal*, 17(1):184–200, 2013.

[46] Brian Van Essen, Roger Pearce, Sasha Ames, and Maya Gokhale. On the role of NVRAM in data-intensive architectures: an evaluation. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2012.

[47] J. S. Vetter and S. Mittal. Opportunities for nonvolatile memory systems in extreme-scale high performance computing. *Computing in Science and Engineering*, 17(2):73–82, 2015.

[48] Corrado Villa, Duane Mills, Gerald Barkley, Hari Giduturi, Stefan Schippers, and Daniele Vimercati. A 45nm 1Gb 1.8 V phase-change memory. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 270–271. IEEE, 2010.

[49] Gwendolyn Renae Voskuilen, Alfredo Gimenez, Ivy Peng, Shirley Moore, and Maya Gokhale. Milestone M1 report: HBM2/3 evaluation on many-core CPU WBS 2.4, milestone ECP-MT-1000. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States); Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2018.

[50] Young Jun Yoon, Byung Deuk Jeon, Byung Soo Kim, Ki Up Kim, Tae Yong Lee, Nohhyup Kwak, Woo Yeol Shin, Na Yeon Kim, Yunseok Hong, Kyeong Pil Kang, et al. 18.4 An 1.1 V 68.2 GB/s 8Gb Wide-IO2 DRAM with non-contact microbump I/O test scheme. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 320–322. IEEE, 2016.

[51] Cristian Zambelli, Gabriele Navarro, Véronique Sousa, Ioan Lucian Prejbeanu, and Luca Perniola. Phase change and magnetic memories for solid-state drive applications. *Proceedings of the IEEE*, 105(9):1790–1811, 2017.

[52] Wangyuan Zhang and Tao Li. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pages 101–112. IEEE, 2009.

[53] Pin Zhou, Vivek Pandey, Jagadeesan Sundaresan, Anand Raghuraman, Yuanyuan Zhou, and Sanjeev Kumar. Dynamic tracking of page miss ratio curve for memory management. In *ACM SIGOPS Operating Systems Review*, volume 38, pages 177–188. ACM, 2004.

Appendix A
Artifact Description Appendix: Siena: Exploring the Design Space of Heterogeneous Memory Systems

A. Abstract

This artifact provides information to generate the computational results that are presented in the paper titled Siena: Exploring the Design Space of Heterogeneous Memory Systems. It includes instructions for installing the software, configuring the workflow of experiments and also scripts for visualizing the results. We also provide job scripts for launching batch sweep tests on Stampede supercomputer at TACC.

B. Description

1) Check-list (artifact meta information):

- **Algorithm: data-centric abstraction of workloads**
- **Program: C/C++**
- **Compilation: GCC version above 5.4.0 (with support for c++11)**
- **Binary: DRAMSim2, Ramulator, and NVDIMMSim libraries**
- **Run-time environment: tested on GNU/Linux 3.10.0 and CentOS 7.3**
- **Hardware: Stampede (SKX nodes) at TACC, ExCL (Intel Xeon E5-2683) at ORNL**
- **Run-time state: display periodic progress and states to the console**
- **Execution: execution time scales with the number of enabled memory simulators**
- **Output: the statistics of caches, the statistics of each memory simulators, and the statistics of each executed kernel**
- **Experiment workflow: Install DRAMSim2, Ramulator, and NVDIMMSim libraries, download the software source code, compile the software, customize experiments, execute experiments, and visualize results**
- **Experiment customization: Yes**
- **Publicly available?: Yes**

2) How software can be obtained (if available): Please download the source code from the git repository at https://github.com/ipftg/siena

3) Hardware dependencies: It compiles and runs on general x86 processor. Use multi-core processors with large main memory for exploration experiments.

4) Software dependencies:

- Autoconf
- GNU Bison
- ASPEN DSL (version 2.0 included in this package)
- C++ compiler with support for C++11 standard
- DRAMSIM2 library binary
- Ramulator library binary
- NVDIMMSim library binary

5) Datasets: The package includes application models and machine configurations for experiments.

C. Installation

The instructions for compiling the dependency libraries and the software are presented as follows.

1) Compile DRAMSim2 simulator into binary libdramsim.a and then update the variable DRAMSIM_ROOT in Makefile.

```
git clone https://github.com/umd-memsys/
    DRAMSim2.git
make libdramsim.a
```

2) Compile Ramulator simulator into binary libramulator.a and then update the variable RAMULATOR_ROOT in Makefile.

```
git clone https://github.com/CMU-SAFARI/
    ramulator.git
make libramulator.a
```

3) Compile NVDIMMSim simulator into binary libnvdsim.a and then update the variable NVDIMMSIM_ROOT in Makefile.

```
git clone https://github.com/
    jimstevens2001/NVDIMMSim.git
make libnvdsim.a
```

4) Compile the Aspen library (GNU Bison and Autoconf are required, Python 2.7 is optional) and then update the variable ASPEN_ROOT in Makefile.

```
cd aspen
autoconf
./configure --prefix=YOUR_TARGET_PATH
make aspen
```

5) In the root directory, configure the level of verbose by changing compilation flags in Makefile. After compiled successfully, run 'make test' to launch a test suite.

```
make
make test
```

D. Experiment workflow

1) Prepare application abstractions. We provide a list of application abstractions in /models/applications. For each application, scale up the problem sizes by changing the scaling parameters as described in Table II. A script that automatically generates application abstractions for Figure 5a and 5b can be found in /scripts/generate_app.sh. Execute:

```
cd ./scripts
./generate_app.sh app_name min_size
    max_size [target_folder]
```

2) Prepare machine models. Figure 4a and 4b sweep a list of vertical architecture options that are defined in /models/machines/V_HBM_x_DDR_y.aspen. Figure 6a and 6b sweep a list of horizontal architecture options that are defined in /models/machines/H_HBM_x_DDR_y.aspen and H_HBM_x_NVM_y.aspen. A script that automatically sweep architecture options can be found in /scripts/generate_arch.sh. Execute:

```
cd ./scripts
./generate_arch.sh arch_type=V|H device0
     size0 [deviceN sizeN] [
    target_folder]
```

3) Prepare mapping models used in Figure 6a and 6b. We provide a script to automatically generate all possible combinations of mapping. The experiments only select those mappings in Table III to execute. Execute:

```
cd ./scripts
./generate_mapping.sh input_app.aspen
    input_arch.aspen [target_folder]
```

4) Prepare job scripts and then launch sweep tests in batch. For memory simulations, place memory specification files in /models/machines/configs and then export to environment variables DRAMSIM_ROOT, RAMULATOR_ROOT and NVDIMMSIN_ROOT. To expedite experiments, we provide scripts to launch multiple experiments in parallel on Stampede supercomputer at TACC. To reuse the job script, simply update the lists defined in applist, machlist, and maplist.

```
#SBATCH −J sweep
#SBATCH −p skx−normal
#SBATCH −N 1
#SBATCH −n 1
#SBATCH −t 01:00:00

#!/bin/bash
DIR:=${CURDIR}
EXE_DIR=$DIR/bin
MACH_DIR=$DIR/models/mach
APP_DIR=$DIR/models/app

declare −a applist=(
”app0”
”app1”
”app2”
)
appnum=${#applist[@]}

declare −a machlist=(
”mach0”
”mach1”
”mach2”
)
machnum=${#machlist[@]}

declare −a maplist=(
”mapping0”
”mapping1”
”mapping3”
)
mapnum=${#maplist[@]}

export RAMULATOR_ROOT=${MACH_DIR}/config
export DRAMSIM_ROOT=${MACH_DIR}/config
export NVDIMMSIM_ROOT=${MACH_DIR}/config

for (( i=0; i<${appnum}; i++ ))
do
  APP=${applist[$i]}
  for (( j=0; j<${machnum}; j++ ))
  do
    MACH=${machlist[$j]}
    for (( k=0; k<${mapnum}; k++ ))
    do
      MAP=${maplist[$k]}
      echo ”Exec ”${APP}” on ”${MACH}”
          by mapping ”$MAP
      ${EXE_DIR}/main ${APP}.aspen ${
          MACH}.aspen ${MAP}.aspen >
          output_${APP}_${MACH}_${MAP}.
          out 2>&1 &
    done
  done
  wait
done
exit
```

5) Extract results from the output files and then visualize the results with MATLAB. The scripts used to generate the plots can be found in /plots/sweep*.m. A script that automatically extract the elapsed cycles from execution can be found in /scripts/generate_result.sh. Execute:

```
cd ./scripts
./generate_result.sh
```

E. Evaluation and expected result

The expected experiment results should include the size of total allocations, number of operations, the elapsed cycles, the statistics of each cache levels, and the statistics of each simulated memory in the system. In addition, the performance results include the profiled execution time.

F. Experiment customization

The framework can be easily extended to integrate with new memory simulators by adding two files ar_simulatorNew.h and ar_simulatorNew.cpp. The new file should implement the interface that is defined in ar_memsim.h.

G. Notes

More details about installation and configuration can be found in README.