Avargues *et al. Journal of Big Data* (2023) 10:75
https://doi.org/10.1186/s40537-023-00761-0

Journal of Big Data

RESEARCH Open Access

Main memory controller with multiple media technologies for big data workloads

Miguel A. Avargues[1*], Manel Lurbe[1], Salvador Petit[1], Maria E. Gomez[1], Rui Yang[2], Xiaoping Zhu[2], Guanhao Wang[2] and Julio Sahuquillo[1]

*Correspondence: miavgu@inf.upv.es

[1] Dep. de Informática de Sistemas y Computadores, Universitat Politècnica de València, València, Spain
[2] Huawei Technologies Co. Ltd., Shenzhen, People's Republic of China

Abstract

SRAM and DRAM memory technologies have been dominant in the implementations of memory subsystems. In recent years, and mainly driven by the huge memory demands of big data applications, NVRAM technology has emerged as a denser memory technology, enabling the design of new hybrid DRAM/NVRAM memory hierarchies that combine multiple memory media technologies to balance memory capacity, latency, cost, and endurance. Two main approaches are being applied to the design of hybrid memory hierarchies: the hybrid address space approach, which relies on the programmer or the operating system to choose the memory technology where each memory page should be stored; and the (only) NVM address space approach, where a faster technology (e.g. commodity DRAM) is needed to acts as a cache of NVRAM to boost the performance. This approach presents architectural challenges such the organization of metadata (e.g. cache tags) and the selection of the proper technology for each memory component. In contrast to existing approaches, this work proposes a memory controller that leverages novel memory technologies such as eDRAM and MRAM to mitigate NVRAM bus contention and improve the performance of the NVM address space. The devised solution proposes a two-level cache hierarchy in the memory controller: a SRAM sector cache and a (x)RAM cache. The (x)RAM cache, much denser, helps significantly reduce the number of accesses to NVRAM. Experimental results show that implementing the (x)RAM cache with eDRAM or MRAM is the best performing approach. Moreover, the eRAM is able to improve the SRAM cache miss penalty by up to 50% and 80%, and overall system performance by 15% and 23%.

Keywords: gem5 simulator, NVMain simulator, Main memory, NVRAM media

Introduction

Memory hierarchy [4] has played a key role in the system performance since earliest processors. The first microprocessors only implemented the RAM main memory level, and secondary memory like tapes and disks. At the end of the 70's, cache memories [34] were introduced to reduce the average main memory access time. Cache memories reduce this time thanks to exploiting localities, spatial and temporal, that data exhibit [2, 32]. Temporal locality refers to the same data being accessed again soon along the execution time, and spatial locality refers to data located in neighboring memory addresses are likely to be accessed soon. As the gap between the processor speed and main memory

has grown, multiple levels of cache memories have been introduced, known as cache hierarchy. Current processors usually implement three on-chip cache levels.

SRAM and DRAM technologies have been used in the past as the dominant memory technologies. SRAM is faster and has been commonly used in the upper levels of the hierarchy to implement on-chip caches. In contrast, the much denser but slower DRAM technology has been used for main memory where more capacity is required as this technology presents reduced cost per GB compared to SRAM. Main memory performance has a big impact on the processor performance as a key role of this memory is to hold most of the data -working set- being used by the running applications. On a (page) miss in the main memory, an access to the secondary storage, several order of magnitude slower, is needed to bring the missing page, which will translate into important penalties in the overall system performance.

Much work has been published in the past to address the well-known memory wall [35, 39]. However, over the last decade two main trends have appeared that have called for revisiting the main memory design. On the one hand, the fact of the increasing core count in multicore processors implies more memory accesses compete for main memory access; therefore, there is a need of increasing both capacity and bandwidth to allow increasing memory level parallelism (MLP) and serving more requests in parallel. In other words, the memory wall continues to grow, especially in huge memory demands big data workloads whose needs already overpass multiple TB in current applications [25], which exceeds the memory memory capacity of existing commodity processors.

To deal with these trends the DRAM memory subsystem has significantly improved in terms of storage capacity and memory bandwidth with each new generation. DRAM has scaled (in Mbits/chip) in a 2× factor every 1.5 years from 1985. However, this trend has slowed down since the beginning of this century and becomes challenging below 10nm technology node since the implementation becomes unreliable [23, 27]. In addition, current server processors use to implement several multi-channel memory controllers to allow supporting higher MLP.

Consequently, conditions have changed as new memory technologies like NVRAM (Non-Volatile RAM) [7] have emerged in the last years, both denser and more energy-efficient than DRAM. The main goal is to used them to implement much bigger main memories [12]. Nevertheless, they present a major downside as they are slower and present finite write endurance [14, 20, 29, 44]. Because each technology presents its advantages and downsides, computer architects can focus on the design of new off-chip memory hierarchies by using multiple technologies, what is known as hybrid main memories. The design pursues to take *the best of each world*; for instance, using DRAM for speed and NVRAM for higher storage capacity. There is an open design space where computer architects play a key role mingling distinct memory technologies to balance: capacity, latency, cost and endurance.

In summary, the huge demands of main memory space, some exceeding 1TB, open an era for computer architects to reach key memory goals, that is, scalability, endurance, and energy consumption. To this end, the paper focuses on a novel main memory design that combines multiple technologies.

Unlike existing approaches, the proposed solution consists on implementing a two-level cache hierarchy combining distinct technologies in the memory controller. This

way attacks the performance bottleneck of the slow accesses to the NVRAM, mitigating NVRAM bus congestion and improving performance.

Problem definition

Performance of emerging applications, such as those of the big data area, can severely suffer when the physical main memory space is not able to accommodate their working set, and consequently, the several order of magnitude slower secondary memory (e.g. solid state drive or hard disk drive) needs to be accessed.

The previous rationale means it is important to face the problem of providing enough memory capacity in order to sustain the performance of these applications. Moreover, as memory demands of big data applications continue to grow exponentially, the memory wall problem is expected to aggravate in the near future as much larger main memories will be needed that cannot be met with existing memory hierarchies. This increasing memory demands was foreseen so that it motivated significant technology advances for the last decade in the development of alternative technologies to DRAM like non-volatile memories, as well as their recent introduction in the processor market [1, 13, 41].

Nowadays, we have entered in an era where multiple technologies have already emerged, each one presenting its advantages and shortcomings. It is a job of the computer architects to design new memory controllers able to mingle multiple technologies to address performance and technological issues like endurance, fault tolerance and energy. The memory controller cannot be already seen as a device that controls a single memory media (e.g. DRAM). Instead, multiple types of media will need to be taken into account to face the mentioned technological and performance issues.

A plausible approach to illustrate the problem is using only a small subset of technologies, e.g. DRAM and NVRAM [28]. The former faster but less denser than the latter. The problem can be addressed following two main approaches: (i) *NVRAM only* address space, and (ii) *Hybrid* –DRAM plus NVRAM– address space,

In the *NVRAM only* approach, as NVRAM presents larger storage capacity, DIMMs in this technology (i.e. NVDIMMs) could be only used to built the main memory; thus, completely replacing current commodity DRAM DIMMs as the main memory device [15–17]. In this way the processor address space would be delimited by the storage capacity of the NVDIMMs. However, important technological challenges appear due to the high write access latency and limited endurance of NVRAM as well as performance issues.

In contrast, in the *Hybrid* approach, the NVRAM could be placed into the memory hierarchy alongside DRAM to augment its address space [9, 19, 22, 37, 38, 40, 42]. In this approach, the processor address space consists of the sum of the storage capacity of both memory media. The main downside of this approach is how the target location (DRAM or NVRAM) of memory pages is selected at execution time. More precisely, as DRAM is faster than NVRAM, the execution time of a program can be accelerated by placing the most accessed pages in this memory media. However it is not done automatically but this approach relies on the programmer or the operating system to choose the main memory media where each memory page should be stored. More precisely, a major drawback of this type of implementation is the complexity it imposes on the application programmer, the operating system, or the corresponding system libraries. Furthermore,

the differences between NVRAM and DRAM memory capacities, which vary by two or three orders of magnitude, make it difficult to design a fast and efficient hybrid memory subsystem. In addition, this approach brings little performance gains when the addressable space of the NVRAM is much bigger than that of the DRAM, which is expected to be the future trend as DRAM scalability imposes many technological challenges [33].

Finally, the design imposes many challenges from an architectural point of view like (i) as tags require a large storage capacity in case of a commodity DRAM is used as a cache, how tags are implemented? (ii) which technologies should be used, where and why? (iii) is there a need of implementing a big cache in the memory controller to allocate data from the different media? In such a case, a sector cache would help to boost performance? (iv) as DRAM read is destructive, how this fact will affect the definition and the memory operation?

Existing solutions

Existing solutions rely on hybrid memory systems built with DRAM and NVM technologies, and can be grouped according to the way the main memory address space is organized: hybrid address space or NVM address space.

Hybrid address space: in these solutions, the NVMRAM and the DRAM share the same address space, that is, each of them stores a different subset of the physical memory pages. This approach is followed in some body of works [5, 6, 9, 11, 24, 30, 31, 38, 43], however, requires modifications both in the memory controller and in the operating system. Management policies in this hybrid memory subsystem can significantly affect the system performance. Therefore, significant effort is required to carefully optimize such policies. The work in [9] proposes to combine PRAM (phase change random access memory) and a low overhead hybrid hardware-software solution for managing the hybrid memory performing the page swapping/migration. Panthera [38] proposes combining NVRAM and DRAM in the context of big-data processing and a memory management technique for efficient data placement and migration. The authors in [31] propose an analytical Markov-based model to estimate the performance and lifetime of DRAM-NVM hybrid memories on various workloads. The proposed model estimates the hit ratios and lifetime for various configurations of DRAM-NVM hybrid main memory. Such an analytical model can aid designers to tune hybrid memory configurations to improve performance and/or lifetime. The work in [19] proposes utility-based hybrid memory management (UH-MEM), a new page management mechanism for various hybrid memories, that systematically estimates the utility (i.e., the system performance benefit) of migrating a page between different memory types, and uses this information to guide data placement. The authors of [30] propose a hybrid memory design jointly with a hardware-driven page placement policy. The policy relies on the memory controller (MC) to monitor access patterns, migrate pages between DRAM and PCM, and translate the memory addresses coming from the cores. Periodically, the operating system updates its page mappings based on the translation information used by the MC.

NVM address space: solutions in this approach organize the hybrid main memory in a hierarchical way, where the NVM occupies the lower level, thus the NVM has the entire main memory address space and the DRAM acts a cache of the NVM [22, 42]. The work in [42] proposes a new caching policy that improves hybrid memory performance and

energy efficiency. The policy tracks the row buffer miss counts of recently used rows in PCM, and caches in DRAM the rows that are predicted to incur frequent row buffer misses. This work uses commodity DRAM and caches entire rows of NVRAM memory banks in DRAM. The proposal in [22] introduces efficiently managing the metadata (e.g., tags) for data cached in DRAM at a fine granularity. This paper uses a small buffer to just cache on-chip the metadata for recently accessed rows. The proposal also adapts dynamically to choose the best data granularity.

NVM address space: approaches are more practical than *Hybrid address space* as they require no software changes, and are closer to our work; therefore, our comparison only focuses on *NVM address space* approaches. The described approaches present important constraints that are being overpassed by our approach. In this regard, our proposal implements a much smaller cache than the proposed in [42] that can be integrated within the same processor package or even on the same die, assuming the selected cache technology is CMOS-compatible. In addition, the cache requirements in our proposal are relatively much lower than those of [22], which eliminates the need for special handling of metadata and allows us to implement a smaller cache with alternative technologies.

Proposed solution

This section presents the proposed solution to address the huge memory requirements of big data applications and to face the major challenges discussed that a multiple media memory controller imposes.

The major memory components of the memory controller board are the media controllers, the NVRAM, the DRAM and an on-board cache. The NVRAM [12, 18] is denser and slower than the DRAM, and it will be used as the main memory address space. The on-chip cache hierarchy is expanded off-chip through the memory controller board. This small board contains the multiple controllers for each media, e.g. for DRAM and NVRAM. We keep the DRAM or commodity DRAM but it will be used as a (second level on-board) cache of the NVRAM. In addition, a smaller but faster sector cache built with SRAM technology is introduced between the on-chip caches and the memory media, to aid in reducing the slowdown. The cache is implemented as a sector cache to serve as a prefetcher when accessing the slower memory media. In this way the long latencies of the slower memory media are amortized as a sector consists of multiple (4 in our design) typical 64B cache blocks.

The design proposed in this paper is based on two important observations that we found during the design process: i) the DRAM cache is often accessed, and ii) a relatively small size (compared to normal sizes of commodity DRAM) seems to be adequate to catch an important fraction of the NVRAM accesses. These results motivate us to focus the research on the usage of alternative memory technologies such as Magnetic Random Access Memory (MRAM) [3, 8] or embedded DRAM (eDRAM) [10, 21], which are faster and consume less energy than commodity DRAM.

The devised approach implements four major components: the HMC Sector Cache or simply Sector Cache, the DRAM/eDRAM/MRAM media, the TAG Cache, and the NVRAM media to achieve the mentioned goals. Next we elaborate the design of these components.

Elaboration

Challenges and major design issues

A major challenge when using commodity DRAM as a cache is where the tag array is placed. If it is placed on the same DRAM DIMM, an access needs to be done to look up whether the target block is in the cache. As DRAM is destructive and relatively slow, this means that a precious time can be lost regardless of whether the target sector is or is not in the DRAM DIMM. To avoid this shortcoming, an alternative approach could be to implement it in an on-board cache (i.e. placed in the memory controller board). However, this design is challenging mainly due to the high capacity (tens of GB) of recent DRAM DIMMs.

To address the mentioned challenges, we explored smaller in-house DRAM designs. The key idea was to reduce the tag array in order to implement it in an SRAM cache. We found that there is no need to use big commodity DRAM DIMMs, but smaller *in-house* DRAM memories can be built and, therefore, their tag array can by implemented in a much faster SRAM cache. We refer to as data cache and tag cache, to the caches keeping the data array and tag array respectively. Moreover, under these assumptions, a direct-mapped cache is able to achieve good performance because of conflict-misses rarely appear with a relatively large DRAM cache.[1]

The question that arises is how large should be the data cache. In principle, the cache should be built with a technology dense enough to store a large amount of data. In this paper, we first explored commodity DRAM (DRAM Cache). However, as our experimental results will show, this DRAM cache is frequently accessed and a relatively small size is sufficient to capture a significant portion of NVRAM accesses.

We improve the performance of the memory controller with two main design options over existing solutions. On the one hand, the fact that the DRAM cache is often accessed means that the main performance gains will come from speeding up this cache. On the other hand, as a relatively small cache is needed, other technologies either CMOS compatible (e.g. eDRAM) or denser (e.g. MRAM), can be used to be integrated in the memory controller board. In this way, important hardware like the memory channel and DIMM's socket can be saved as well as energy consumption over using commodity DRAM DIMMs. Therefore, in this work we consider MRAM, eDRAM, and in-house DRAM memory tailored for the needs of the proposed controller.

The design of the basic memory controller consists of two main steps: i) to define the individual components that will compose the memory controller, and ii) to specify the functionality of each component as well as how they interact with each other. After that, iterative refinements need to be done for performance enhancements.

- *HMC Sector Cache or simply Sector Cache*: implemented in the HMC chip, it allows storage of multiple data blocks (known as sectors) in a single cache line. It is modeled as a sector cache to help reduce NVRAM latency as one, whole line sector-sized access is issued instead of several smaller, block-sized accesses. In other words, it acts as a sequential prefetcher by design what helps hide the slow NVRAM media. As a conventional sector cache, each line contains metadata for each individual block

[1] This claim was checked with the experimental framework and workloads studied in this paper.

Table 1 Comparison of the considered memory technologies

	MRAM	eDRAM	In-house DRAM	Commodity DRAM
Speed	−	++	+	− −
Energy consumption	− −	−	+	++
Durability	− −	−	+	++

For each feature, four levels are defined: "++", "+", "−", and "− −"

(dirty and valid) and for the whole line (present and dirty). This cache is the first structure that is checked when a LLC requests reaches the HMC. Therefore, it is expected to bring important performance benefits.

- *DRAM/eDRAM/MRAM media*: in the initial controller design, typical DRAM media is used to provide a second storage cache level in order to reduce average latency of accessing the NVRAM media. In other words, with the DRAM cache, the faster (but smaller) DRAM media is accessed instead of the NVRAM. This not only has latency benefits, but may help with NVRAM wear as well. In addition, we also explore MRAM, eDRAM media and in-house DRAM. Table 1 compares the considered technologies based on their speed, energy consumption, and durability. Each of these features is classified into four different levels. In terms of speed, MRAM, eDRAM, and in-house DRAM surpass traditional DRAM as they can be integrated within the same package as the processor. Among these technologies, eDRAM is the fastest due to its CMOS compatibility, allowing it to be implemented on the same die as the controller, followed by in-house DRAM and finally MRAM, with a slower write speed. MRAM also has the lowest energy consumption as it does not require refresh. In this regard, the eDRAM technology presents the lowest energy consumption among the DRAM technologies, followed by in-house DRAM and then commodity DRAM, whose energy consumption is negatively impacted due to its off-chip implementation. Lastly, with respect to durability, MRAM presents the weakest performance because of its limited write cycles. For dynamic memory technologies, their durability is primarily determined by the capacitors' size, putting eDRAM below DRAM.
- *TAG Cache*: together with the DRAM/eDRAM/MRAM media - which stores the data blocks - an (x)RAM Cache is formed. Accessing directly to this cache upon every sector cache miss would result in significant refreshing overhead (for dynamic memory media) and a high number of unfruitful accesses (i.e. the data block/sector is not in the x-RAM cache but in the NVRAM media). To deal with these shortcomings, we use SRAM technology to provide fast access time to the tags. That is, both structures are accessed sequentially, so that in case of a tag miss (meaning that the block is stored in NVRAM media) the DRAM is not accessed (so avoiding unfruitful accesses to the slow DRAM media). This design choice provides fast access to the tag array in case the data access is slow and has been already used in the past in modern processors. For instance, the IBM POWER4 [36] deploys the L3 data array in an individual chip different from the processor chip. To avoid unnecessary accesses and provide fast access to the tag array in order to discern whether the L3 or the main memory should be accesses, the tag array is implemented in the processor chip. Usu-

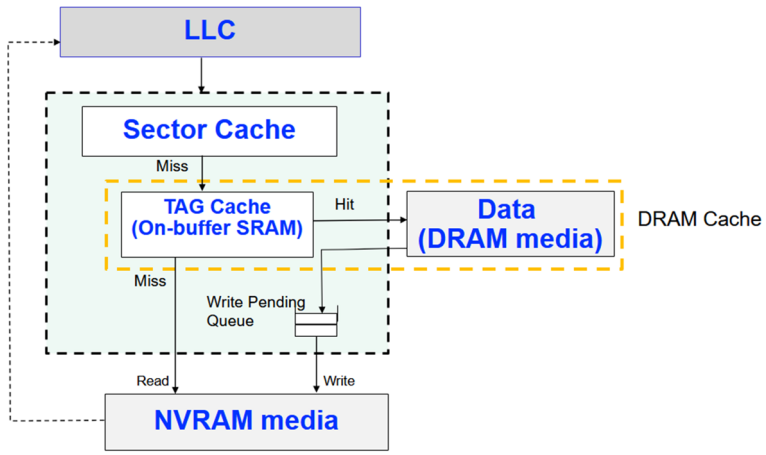

Fig. 1 HMC Controller components diagram

ally, designing tags for a commodity DRAM cache is challenging mainly due to the high capacity (tens of GB) of recent DRAM DIMMs. The problem, however, is much more affordable or it is not an issue at all for smaller cache sizes. As we found that there is no need to use so a huge cache, but a much smaller is enough, the challenge becomes much more affordable. Indeed, under these assumptions, a direct-mapped cache already achieves good performance because of the cache size conflict-misses rarely appear. This is because the huge size differences between the cache and the main memory, which makes very unlikely that two different sectors are mapped to the same cache line. This claim was checked with the experimental framework and workloads studied in this paper.

- *NVRAM media*: contains the DIMMs with NVRAM technology.

Figure 1 depicts the devised components and the up-down data flow. We would like to emphasize that this figure shows only the major components for the devised baseline HMC approach. The auxiliary components (such as stream buffers required for the prefetching technique currently being worked on) will be added as the project advances during the development of the HMC Component.

Simulation framework

We are now going to introduce the simulator framework used to carry out all the test executions. This environment is built around two main simulators, gem5 and NVMain. The former has been used to model the processor side of the system, the latter to model the main memory subsystem. NVMain can be made to compile as a gem5 extra, which means that in only one binary we can use both gem5 21.0.1.0 and NVMain. Next, we summarize the main features of each one.

The gem5 simulator is an event-driven modular simulator. This simulator offers great flexibility in the system configuration, thanks to the use of Python scripts that initialize the system components and act as the source of system configuration. This allows for complex processor and cache hierarchy configurations. It also has two distinct working modes, full system simulation (FS) and system-call emulation (SE). In FS mode, gem5

simulates a whole physical system from boot, similarly as a real machine. On the other hand, SE mode offers a fast way to simulate a binary under a different system configuration of the real machine that is running the simulation.

NVMain is a cycle accurate main memory simulator that models both non-volatile and DRAM memories. In addition to being used as an extra for gem5, NVMain can be compiled standalone, which allows to simulate the memory subsystem using traces. NVMain allows for the implementation of components of the main memory system, through the extension of base classes defined in C++. It also offers support for hybrid memory systems, which is a key feature in this project. Also, the memory model can be modified via the use of configuration files, such as timing parameters, memory geometry or memory controller.

Modeled components and simulation tools

To evaluate our approach we built the whole system, ranging from the processor core to the lower levels of the memory hierarchy, and passing through the cache hierarchy levels.

The gem5 simulator was used to model the processor side including the processor core, L1 and L2 caches, and the HMC sector cache. No L3 cache was modeled since our main focus is on the main memory subsystem and not on the processor package. Due to this reason, we keep the L2 cache large enough to catch most of the memory accesses as in real processors that include an L3 cache. The HMC (sector) cache was also modeled in gem5. As NVMain is a specific memory simulator it was used to implement the main memory controller that drives the distinct memory media (e.g. DRAM or NVRAM). Figure 2 presents the major components of the system in a hierarchical way.

The memory controller keeps the media controllers of the attached memory devices, as well as the tag array of the commodity DRAM media that was modeled as implemented in SRAM for performance. As mentioned above, alternative media like eDRAM or MRAM have been modeled.

System parameters

To assess our approach, the system configuration parameters shown in Table 2 were used. Access times (latency) were obtained with CACTI [26] in ns for a 22 nm technology node and translated to processor cycles for a 2GHz processor frequency. The size of the HMC cache and the DRAM cache have been scaled down to the problem size of the workloads to obtain representative values. Using this tool, we were able to model the different memory access time using as a base memory a 4GB 2933-DDR4 DIMM, which is the memory we refer to as "commodity DRAM". As for the in-house DRAM, we scaled down the commodity DRAM from 4GB down to 64MB. The eDRAM technology was modeled as a "cache" memory type, instead of "main memory" in the CACTI configuration. Finally, MRAM access values were modeled according to HUAWEI Technologies (personal communication).

For evaluation purposes, memory timing parameters should be representative presenting those hardware level details with significant impact on performance. For instance, when the required data is located in an open memory page (i.e., row buffer hit), the access time is about three times smaller as if the target data is not (i.e., row

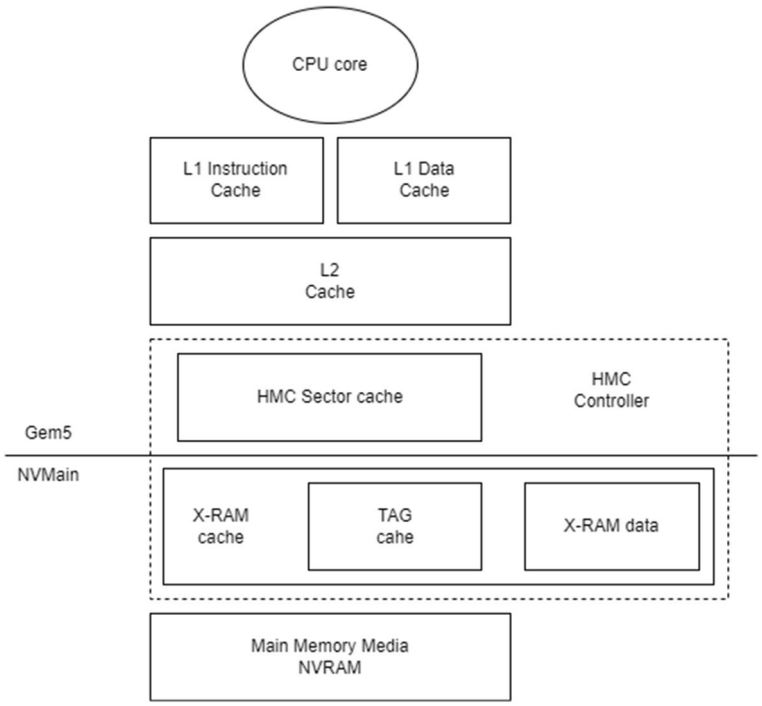

Fig. 2 Simulation tools used and system components we modeled in each of them. The gem5 simulator in the upper side and NVMain in the lower side of the figure. In case of commodity DRAM the X-RAM Data is outside the HMC controller bound

Table 2 gem5 system configuration parameters for a 22 nm technology node

Component	Parameter	Value
CPU	Type	TimingSimpleCPU
	Frequency	2.0 GHz
L1 Data cache	Size, block size, associativity	32 KB, 64 B, 8-way
	Tag/Data latency	3/3 cycles
L1 Inst. cache	Size, block size, associativity	32 KB, 64 B, 8-way
	Tag/Data latency	3/3 cycles
L2 Cache	Size, block size, associativity	2 MB, 64 B, 16-way
	Tag/Data latency	11/11 cycles
	Write policy	Write-back
HMC Sector cache	Size, block size, associativity	8 MB, 256 B, 16-way
	Tag/Data latency	17/17 cycles
	Write policy	Write-back
x-RAM cache	Memory channels	2
	Total space storage	64MB

Cycles have been calculated for a 2 GHz frequency

buffer miss), and thus, the precharge and active commands needs to be send by the memory controller. Table 3 summarizes the main features of the studied technologies to assess our approach in memory bus cycles. Access times in case of both row buffer miss (RB miss) and row buffer hit (RB hit) are shown. The table also specifies the values for the row precharge delay (t_{RP}), row to column delay (t_{RCD}) and column

Table 3 Memory media specifications

	Comm. DRAM	In-house DRAM	eDRAM	MRAM	NVRAM
Tag lookup (ns/cc)	4/8				0
Read access time (ns)					
RB miss (ns)	43	16.5	10	10	176.2
RB hit (ns)	14	5.5	3	3	43
Write access time (ns)					
RB miss (ns)	43	16.5	10	50	176.2
RB hit (ns)	14	5.5	3	16	43
Memory frequency	3800 MHz				488
Processor frequency	2000 MHz				
cc (RBmiss/RBhit)	86/28	33/11	20/6	20/6R 100/32W	353/86

CC refers to core cyles

latency (*CL*). The bottom row quantifies both RB miss and RB hit in core cycles (cc) for a 2GHz core. It is important to note that these access times are a worst case (RB miss) and a best case (RB hit) scenario, as the average access time will depend on the row buffer hit ratio. On the one hand, for applications with high spatial locality the RB hit ratio will be high, thus lowering the access time towards the RB hit access time. On the other hand, for applications with low spatial locality, the RB hit ratio would be rather poor, thus increasing the access time to that of the RB miss latency.

Studied workloads

Three main workloads have been used to evaluate the proposal; below we summarize the main characteristics of each of them.

Redis: this workload is an in-memory storage structure which can be used as a database, cache or message broker. Redis offers different kinds of data structures such as hashes, lists, sets, ordered sets, etc. This workload natively supports data replication, the LRU replacement policy, and multiple on disk persistence levels. Redis is composed of two main benchmarks: "redis-benchmark" and "memtier_benchmark" discussed below. The former is an utility program included in the Redis installation offers a way to simulate commands as if they were done by N clients at the same time for a total of M requests.

Memtier: despite using the Redis server, memtier is a benchmarking tool to test both Memcached and Redis instances. Some of its features are configurable Read/Write ratio and the ability to choose the data access pattern.

MySQL: currently, MySQL is one of the most known database management systems (DBMS). Its code is open source and provides a great variety of key features such as rollback and fault recovery. It is used by giant tech companies such as Google, Facebook or Twitter. We used the "sysbench" benchmarking tool to perform the experiments with the MySQL server.

Table 4 presents the values of the main parameters used in to obtain the experimental results in the studied workloads.

Table 4 Parameter values used in the studied workloads

Redis		**Memtier**		**MySQL**	
Parameter	Value	Parameter	Value	Parameter	Value
Clients	200	Threads	4	Entries	100000
Req/test	45000	Clients	15	Threads	8
Req. Size	2000 B	Data size	1024	Instances	3
Instances	2				

Table 5 HMC miss latency of the studied benchmarks

Benchmark	**Memory**	**Data cache technology**			
		Commodity DRAM	**In-house DRAM**	**eDRAM**	**MRAM**
Redis	(x)RAM cache (cc)	70	49	41	42
	NVRAM (cc)	256	242	242	242
	AVG latency (cc)	101	76	67	67
MySQL	(x)RAM cache (cc)	89	51	43	44
	NVRAM (cc)	254	242	242	246
	AVG latency (cc)	122	78	69	69
Memtier	(x)RAM cache (cc)	81	53	45	46
	NVRAM (cc)	258	252	255	255
	AVG latency (cc)	109	80	71	72

These values include tag lookup (8 cc), access time, and contention

Experimental results

This section presents and analyzes the experimental results obtained with the system parameters (processor and memory media) and workloads discussed above. Results compare and analyze performance differences among the four memory media used as (x) RAM's *data cache*, that is, commodity DRAM, in-house DRAM, eDRAM, and MRAM.

Memory latency analysis at the HMC

The main goal of our approach is to reduce the HMC miss penalty. Two main HMC latencies (or miss penalties) are distinguishable depending on whether the missing sector hits or misses in the tag cache. In the former case the sector will be served by the (x) RAM cache, and in the latter by the NVRAM DIMM. Therefore, the average latency will depend on the tag cache hit ratio. That is, the higher the hit ratio the more requests will be served by the (x)RAM cache (where x refers to specific technology of the data cache) and thus the average latency will be smaller.

Table 5 presents the latency results for Redis, MySQL, and Memtier in processor cycles. Latencies are broken down depending if there is a hit in the tag cache ((x)RAM cache) latency or a miss (NVRAM latency). Average latencies are presented in both cases. Remember that in case of row buffer hit the latency is smaller than in case of row buffer miss. Comparing the average latency (AVG latency) of these three applications, it can be observed that MySQL is the one presenting the highest value (by 21% and 12% greater than Redis and Memtier, respectively). This is because of this value is strongly related to the (x)RAM hit-ratio, as the bigger the hit-ratio the smaller the latency.

Table 6 Redis benchmark statistics for each technology

Memory structure	REDIS/NV-S-D	commDRAM	DRAM	eDRAM	MRAM
L2 (LLC) cache	L2 Miss latency	51	45	43	43
Sector cache	HMC Hit-Ratio	75.93	75.52	75.32	75.41
	HMC MPKI	1.81	1.83	1.85	1.84
	HMC Miss Latency	101	76	67	67
(x)RAM cache	TAG Hit-Ratio (%)	94.87	94.99	95.00	95.02
	(x)RAM Latency	70	49	41	42
NVRAM DIMM	Read / Write (%)	49.82/50.18	49.85/50.15	49.84/50.16	49.83/50.17
	NVRAM Latency	256	242	242	242

All latency metrics are in core cycles (cc)

Table 7 MySQL benchmark statistics for each technology

Memory Structure	MySQL/NV-S-D	commDRAM	DRAM	eDRAM	MRAM
L2 (LLC) cache	L2 Miss latency	76	57	53	54
Sector cache	HMC Hit-Ratio	57.19	57.48	57.56	56.62
	HMC MPKI	1.95	1.95	1.93	1.97
	HMC Miss latency	122	78	69	69
(x)RAM cache	TAG Hit-Ratio (%)	96.54	96.26	96.46	96.18
	(x)RAM latency	89	51	43	44
NVRAM DIMM	Read/Write (%)	49.90/50.10	49.93/50.07	49.91/50.09	49.93/50.07
	NVRAM Latency	254	242	242	246

Holistic evaluation

This section analyzes the performance just before and just after the sector cache. To this end we show the L2 (the LLC in our processor) miss latency and the HMC miss latency. Two main goals are pursued: i) to check how effective the sector cache is and each individual component of the HMC, and ii) to check to which extend the (x)RAM and NVRAM contribute to the final system performance. For the sake of completeness, and to ease the analysis, we show additional metrics like the HMC hit ratio and the Tag Cache hit ratio (i.e. the hit ratio of the (x)RAM cache), as well as the HMC MPKI (misses per kilo-instructions).

Tables 6, 7, and 8 present the results. It can be appreciated that the performance of the memory controller, quantified in terms of HMC miss latency, significantly varies among applications. This latency is over 100 cc for all the studied applications in the commodity DRAM cache which is the worst performing (x)RAM cache, and between 67 and 80 cc in the remaining (x)RAM technologies. These results show that significant performance gains can be achieved with alternative memory media with respect to using commodity DRAM. For instance, it can be observed that the MRAM media improves the performance of the commodity DRAM by 51% (101/67), 77% (122/69), and 52% (108/71) in Redis, MySQL, and Memtier, respectively.

To assess our approach we will use as baseline a system without the (x)RAM cache as baseline. We will refer to the baseline as NV-S model and to our approach as NV-S-D.

Figure 3 shows the latency improvements in both the L2 (LLC) miss penalty and HMC cache miss penalty. Results are shown for each alternative memory media and

Table 8 Memtier benchmark statistics for each technology

Memory Structure	Memtier/NV-S-D	commDRAM	DRAM	eDRAM	MRAM
L2 (LLC) cache	L2 Miss latency	85	68	63	63
Sector cache	HMC Hit-Ratio	40.74	41.24	41.01	40.98
	HMC MPKI	3.39	3.40	3.45	3.40
	HMC Miss latency	108	80	71	71
(x)RAM Cache	TAG Hit-Ratio (%)	82.77	82.90	83.27	82.91
	(x)RAM latency	81	53	45	46
NVRAM DIMM	Read/Write (%)	49.63/50.37	49.55/50.45	49.64/50.36	49.61/50.39
	NVRAM Latency	258	252	255	255

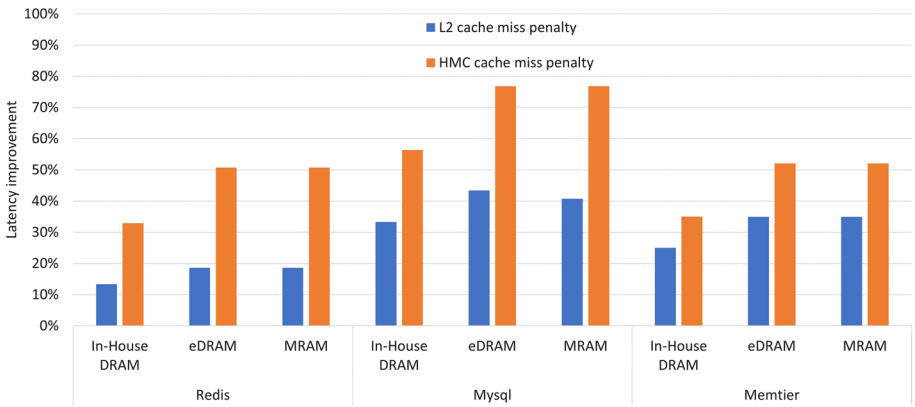

Fig. 3 Latency improvements seen from the processor side (blue) and from the memory controller point of view (orange) over the commodity DRAM memory model

compared over the commodity DRAM that is used as baseline. It can be observed that both eDRAM and MRAM present similar values. Performance gains can be as much as around 77% in the HMC miss penalty but the L2 miss penalty is much lower due to efficiency of the HMC cache.

It is worth noting that the value of the L2 miss latency is not too high as it is always below 85 cycles (see the values in the previous tables). The reason is twofold. On the one hand, the HMC presents a high hit ratio, especially in Redis and MySQL where it is always above 75% and 55%, respectively. On the other hand, the NVRAM is scarcely accessed as the Tag cache of the (x)RAM presents an excellent hit ratio over 82% across all the applications. Moreover, in Redis and MySQL this value is over 94%. This means that despite improving the performance of the memory subsystem, the efficiency of these components makes that little performance gains will be observed in the studied memory models in terms of overall system performance as discussed below.

Analyzing memory latency per component

To analyze to which extend the different components of the memory subsystem contribute to the system performance, we have analyzed their contribution to the AMAT (i.e. average memory access time). To this end, the latencies of each component of the

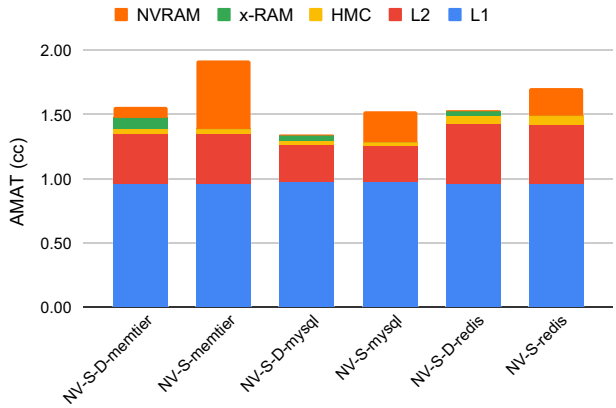

Fig. 4 Per component contribution to the average memory access time (AMAT) in the NV-S and NV-S-D system with AMS_{D4} for the in-house (x)RAM cache

memory hierarchy: L1, L2, HMC, (x)RAM cache and NVRAM are accumulated for all the memory accesses, and then, divided by the total number of memory instructions issued by the processor (i.e. not only those accessing to the HMC). After that, a bar is plotted for the studied systems. Figure 4 presents the results. Two bars are plotted for each benchmark, one for the NV-S-D system and another for the NV-S for comparison purposes.

As expected, the L1 cache (blue slice) presents roughly the same contribution across all the applications, and the L2 cache (red slice) and HMC cache (yellow slice) present the same contribution for the same application regardless of the memory model. It can be observed that the contribution of the HMC is rather low, which means that it is not frequently accessed on average per issued memory instruction. A larger length of the yellow bar means that that application has had a noticeable number of accesses to the HMC and has experienced a high of HMC hit ratio. In the results of the tables presented above, notice that Redis is the application with highest HMC hit ratio (above 75%).

Therefore, to find out the performance differences between the NV-S-D memory model and the NV-S memory model, the comparison relies on comparing the orange bar of the NV-S memory model against the orange plus the green bars of the NV-S-D system since the other bars remain common for a given application. The major performance gains can be appreciated in Memtier as the NVRAM component (orange bar) is almost 3× larger than in the other applications. The results show that the proposal is effectively working across all the applications, as the orange bar is significantly reduced.

The results presented in upper side (orange and green bars) of this plot corroborates the latency improvement results presented in Fig. 3. However, the overall performance depend on how the total bar length is reduced (in %). This means that the mechanisms applied to the HMC to reduce the latency will strongly depend on the *common components of the bar*. In other words, the reduction of the total bar length is limited by the length of the bars ranging from the yellow to the blue one, both included.

To take away:

The overall performance benefits will be limited by the reduction (in %) of the total AMAT bar length. From an analytical point of view the question is: how could

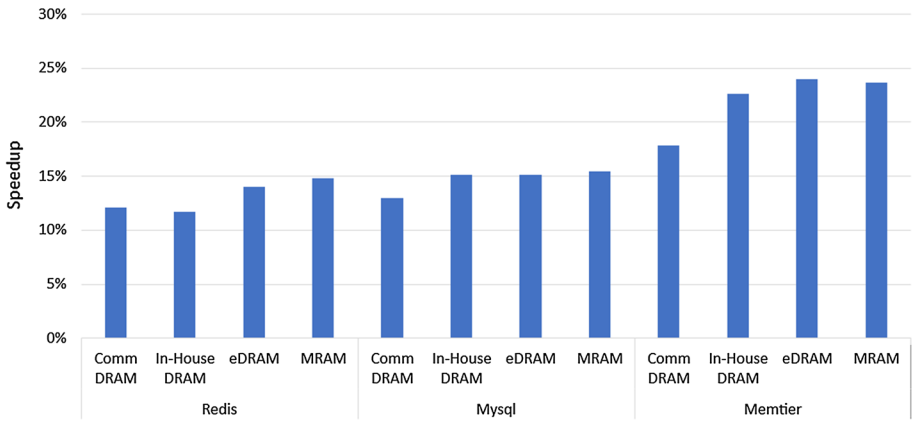

Fig. 5 Speedup in terms of IPC (or execution time) of using alternative memory media over the NV-S system

we improve the AMAT? More precisely, how could we reduce the difference (in %) between both bars? There are two main options: i) reducing the length of the bars below the yellow one or ii) enlarging the length of the bars above the yellow one. The former can be achieved by acting on the processor side, the latter on the HMC side as discussed througout this paper.

Improvements on the system performance

Once we have analyzed the main components and their contribution to the AMAT, we study to which extend latency improvements along the cache and memory hierarchy translates into final system performance improvements. In other words, we analyze the speedup the devised HMC controller introduces.

Figure 5 presents the results for each technology over the NV-S memory model. System performance gains range from around 12% to around 24%. Commodity DRAM offers much poorer performance than the other technologies with the only exception of Redis where similar performance (1% difference) as in-house DRAM can be observed. As expected from the previous studies, eDRAM and MRAM are the best performing ones and achieve similar performance, as much as 15% in MySQL and 24% in MRAM. Remember that these performance gains are achieved by adding the (x)RAM technology, that is, the second level cache, as the NV-S system already includes the HMC cache. Therefore, we can conclude that significant performance gains can be achieved with the eDRAM and MRAM technologies.

Conclusions

This paper has focused on improving the performance of memory systems for big data applications focusing on the memory controller and alternative memory media. Unlike existing approaches, the devised solution proposes a two-level cache hierarchy combining distinct technologies in the memory controller. The key idea is to attack the performance bottleneck that introduces the slow NVRAM, mitigating bus congestion and improving performance.

In-house DRAM, eDRAM and MRAM memory technologies have been analyzed acting as x-RAM cache. Results show that eDRAM and MRAM are the best performing technologies. Compared to the NV-S system, eDRAM and MRAM are able to improve the HMC miss penalty by almost 78% in MySQL and above 50% in the rest of applications. These improvements reduce to around 40%, 33% and 28% when improving the L2 cache miss latency in MySQL, Memtier, and Redis. Overall system performance improves by 15% and 23% depending on the studied workload over using commodity DRAM.

This new design can bring important performance gains in future server processors and cloud systems running big data applications and other applications requiring huge amounts of data.

As any starting design, it needs to be evolved to further improve the overall system performance. In this regard, new research would help such us mechanisms aimed at hiding the large NVRAM latencies or more aggressive processors supporting higher memory level parallelism.

Acknowledgements
The authors would like to express their gratitude to both Huawei Technologies and Spanish Ministerio de Ciencia e Innovación.

Author contributions
MAA: experiments, gathering results and writing. ML: experiments and presenting results. SVP: simulation framework and libraries. MEG: results analysis. RY: new Technologies and results discussion. XZ: new Technologies and results discussion. GW: Huawei project manager and results discussion. JS: main coordinator, writing and result analysis. All authors read and approved the final manuscript.

Funding
This work has been partially supported by Huawei under Agreement No: TC20210705020, by the Spanish Ministerio de Ciencia e Innovación and European ERDF under grants PID2021-123627OB-C51 and TED2021-130233B-C32.

Availability of data and materials
The materials used during development of this paper are available in the https://github.com/miavgu/JBD23_Multiple_Media_Technologies/ repository.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Both parts of the project, Huawei

Competing interests
The authors declare that they have no competing interests.

Received: 30 January 2023 Accepted: 8 May 2023
Published online: 22 May 2023

References

1. Altman A, Arafa M, Balasubramanian K, Cheng K, Damle P, Datta S, Douglas C, Gibson K, Graniello B, Grooms J, et al. Intel optane data center persistent memory. In: IEEE Hot Chips 31 Symposium (HCS). 2019; p. i–xxv.
2. Anghel A, Dittmann G, Jongerius R, Luijten R. Spatio-temporal locality characterization. 1st Workshop on Near-Data Processing (WoNDP). 2013; p. 1–5.
3. Apalkov D, Dieny B, Slaughter JM. Magnetoresistive random access memory. Proceed IEEE. 2016;104(10):1796–830.
4. Balasubramonian R, Jouppi NP. Multi-core cache hierarchies. Berlin: Springer; 2022.
5. Bock S, Childers BR, Melhem RG, Mossé D. Concurrent page migration for mobile systems with os-managed hybrid memory. Computing Frontiers Conference, CF'14, Cagliari, Italy, 2014; p. 31:1–31:10.
6. Bock S, Childers BR, Melhem RG, Mossé D. Concurrent migration of multiple page in software-managed hybrid main memory. In: 34th IEEE International Conference on Computer Design, ICCD, Scottsdale, AZ, USA, 2016;p. 420–3.

7. Burr GW, Kurdi BN, Scott JC, Lam CH, Gopalakrishnan K, Shenoy RS. Overview of candidate device technologies for storage-class memory. IBM Journal of Research and Development, 2008;pp. 449–64.
8. Chen E, Apalkov D, Diao Z, Driskill-Smith A, Druist D, Lottis D, Nikitin V, Tang X, Watts S, Wang S, Wolf SA, Ghosh AW, Lu JW, Poon SJ, Stan M, Butler WH, Gupta S, Mewes CKA, Mewes T, Visscher PB. Advances and future prospects of spin-transfer torque random access memory. IEEE Trans Magnet. 2010;46:1873–8.
9. Dhiman G, Ayoub R, Rosing T. Pdram: A hybrid pram and dram main memory system. In: 46th ACM/IEEE Design Automation Conference, 2009;p. 664–9.
10. Diodato P. Embedded dram: more than just a memory. IEEE Commun Mag. 2000;38(7):118–26.
11. Hassan A, Vandierendonck H, Nikolopoulos DS. Energy-efficient hybrid dram/nvm main memory. In: International Conference on Parallel Architecture and Compilation (PACT), 2015;p. 492–3.
12. Hoya K, Hatsuda K, Tsuchida K, Watanabe Y, Shirota Y, Kanai T. A perspective on nvram technology for future computing system. In: Automation and Test (VLSI-DAT): International Symposium on VLSI Design; 2019. p. 1–2.
13. Intel. 3d xpointTM: a breakthrough in non-volatile memory technology. Intel. 2015. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html?wapkw=3d+xpoint
14. Kargar S, Nawab F. Challenges and future directions for energy, latency, and lifetime improvements in NVMs. Distrib Parallel Databases. 2022. https://doi.org/10.1007/s10619-022-07421-x.
15. Kültürsay E, Kandemir M, Sivasubramaniam A, Mutlu O. Evaluating stt-ram as an energy-efficient main memory alternative. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013;p. 256–67.
16. Lee BC, Ipek E, Mutlu O, Burger D. Architecting phase change memory as a scalable dram alternative. In: 36th International Symposium on Computer Architecture (ISCA), Austin, TX, USA, 2009;p. 2–13.
17. Lee BC, Zhou P, Yang J, Zhang Y, Zhao B, Ipek E, Mutlu O, Burger D. Phase-change technology and the future of main memory. IEEE Micro. 2010;30:143–143.
18. Li D, Vetter JS, Marin G, McCurdy C, Cira C, Liu Z, Yu W. Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium, 2012;p. 945–56.
19. Li Y, Ghose S, Choi J, Sun J, Wang H, Mutlu O. Utility-based hybrid memory management. In: IEEE International Conference on Cluster Computing (CLUSTER), 2017;p. 152–65.
20. Luo Y, Ghose S, Cai Y, Haratsch EF, Mutlu O. Improving 3d NAND flash memory lifetime by tolerating early retention loss and process variation. In: Abstracts of the ACM International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS, Irvine, CA, USA, 2018;p. 106.
21. Matick RE, Schuster SE. Logic-based eDRAM: origins and rationale for use. IBM J Res Dev. 2005;49:145–65.
22. Meza J, Chang J, Yoon H, Mutlu O, Ranganathan P. Enabling efficient and scalable hybrid memories using fine-granularity dram cache management. IEEE Comput Archit Lett. 2012;11:61–4.
23. Meza J, Wu Q, Kumar S, Mutlu O. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In: 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, Rio de Janeiro, Brazil, 2015;p. 415–26.
24. Mogul JC, Argollo E, Shah MA, Faraboschi P. Operating system support for NVM+DRAM hybrid main memory. In: Proceedings of HotOS: 12th Workshop on Hot Topics in Operating Systems, Monte Verità, Switzerland. 2009.
25. Mohammadi Makrani H, Rafatirad S, Houmansadr A, Homayoun H. Main-memory requirements of big data applications on commodity server platform. In: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2018;pp. 653–60.
26. Muralimanohar N, Balasubramonian R, Jouppi NP. Cacti 6.0: a tool to model large caches. HP laboratories, 2009;p. 28.
27. Mutlu O. Rethinking memory system design. In: Mobile System Technologies Workshop (MST), 2016;p. 1–3.
28. Nuns T, Duzellier S, Bertrand J, Hubert G, Pouget V, Darracq F, David JP, Soonckindt S. Evaluation of recent technologies of non-volatile ram. In: 2007 9th European Conference on Radiation and Its Effects on Components and Systems, 2007;p. 1–8.
29. Pelley S, Chen PM, Wenisch TF. Memory persistency. In: 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), 2014;pp. 265–76.
30. Ramos LE, Gorbatov E, Bianchini R. Page placement in hybrid memory systems. In: Proceedings of the 25th International Conference on Supercomputing, Tucson, AZ, USA, 2011;p. 85–95.
31. Salkhordeh R, Mutlu O, Asadi H. An analytical model for performance and lifetime estimation of hybrid DRAM-NVM main memories. IEEE Trans Comput. 2019;68:1114–30.
32. Samdani Q, Thornton M. Cache resident data locality analysis. In: Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728), 2000;p. 539–46.
33. Shiratake S. Scaling and performance challenges of future dram. In: IEEE International Memory Workshop (IMW), 2020;p. 1–3.
34. Smith AJ. Cache memories. ACM Comput Surv. 1982;14:473–530.
35. Sun X-H. Remove the memory wall: from performance modeling to architecture optimization. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, 2006;p. 2.
36. Tendler JM, Dodson JS, Fields JS, Le H, Sinharoy B. Power4 system microarchitecture. IBM J Res Dev. 2002;46:5–25.
37. Van Essen B, Pearce R, Ames S, Gokhale M. On the role of nvram in data-intensive architectures: An evaluation. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium, 2012;p. 703–14.
38. Wang C, Cui H, Cao T, Zigman J, Volos H, Mutlu O, Lv F, Feng X, Xu GH. Panthera: Holistic memory management for big data processing over hybrid memories. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019;p. 347–62.
39. Wulf WA, McKee SA. Hitting the memory wall: implications of the obvious. SIGARCH Comput Archit News. 1995;23:20–4.

40. Xiong A, Bai W, Long L, Jiang Y. Non-volatile memory page allocation mechanism. In: 6th International Conference on Control, Automation and Robotics (ICCAR), 2020;p. 21–6.
41. Yang Y, Cao Q, Wang S. A comprehensive empirical study of file systems on optane persistent memory. In: IEEE International Conference on Networking, Architecture and Storage (NAS), 2021;p. 1–8.
42. Yoon H, Meza J, Ausavarungnirun R, Harding RA, Mutlu O. Row buffer locality aware caching policies for hybrid memories. In: IEEE 30th International Conference on Computer Design (ICCD), 2012;p. 337–44.
43. Zhou P, Zhao B, Yang J, Zhang Y. A durable and energy efficient main memory using phase change memory technology. In: Proceedings - International Symposium on Computer Architecture. 2009;p. 14–23.
44. Zuo P, Hua Y, Zhao M, Zhou W, Guo Y. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2018;p. 442–54. IEEE.

Publisher's Note