Microelectronics Journal 45 (2014) 211–216

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

System-level impacts of persistent main memory using a search engine

Taciano Perez [a,*], Ney Laert Vilar Calazans [b], César A.F. De Rose [b]

[a] HP R&D, Porto Alegre, Brazil
[b] PUCRS, Faculty of Informatics, Porto Alegre, Brazil

ARTICLE INFO

Article history:
Received 2 June 2013
Received in revised form
1 November 2013
Accepted 6 November 2013
Available online 26 November 2013

Keywords:
Non-volatile memory
Persistent main memory
PCRAM
RRAM
Memristor

ABSTRACT

Computer memory systems traditionally use distinct technologies for different hierarchy levels, typically volatile, high speed, high cost/byte solid state memory for caches and main memory (SRAM and DRAM), and non-volatile, low speed, low cost/byte technologies (magnetic disks and flash) for secondary storage. Currently, non-volatile memory (NVM) technologies are emerging and may substantially change the landscape of memory systems. In this work we assess system-level latency and energy impacts of a computer with persistent main memory using PCRAM and Memristor, comparing the development and execution of a search engine application implementing both a traditional file-based approach and a memory persistence approach (Mnemosyne). Our observations show that using memory persistence on top of NVM main memory, instead of a file-based approach on top DRAM/Disk, produces less than half lines of code, is more than 4× faster to develop, consumes 33× less memory energy, and executes search tasks up to 33× faster.

1. Introduction

Since the beginning of mainstream commercial computing, memory hierarchy used in computer design has been employing volatile, high speed memory technologies (SRAM and DRAM) for caches and main memory, and non-volatile, low speed technologies (magnetic disks and flash memory) for secondary storage [1].

Currently, non-volatile memory (NVM) technologies are emerging and may substantially change the landscape of memory systems. Non-volatile memory (NVM) technologies such as phase-change RAM (PCRAM), magnetic RAM (MRAM) and Memristor promise to enable memory chips that are non-volatile, require low energy and have density and latency closer to current DRAM chips [2]. The creation of byte-addressable, non-volatile solid state memory could make a significant amount of persistent main memory available to computer systems, allowing for consolidating these two different levels of the storage hierarchy – main memory and secondary storage – into a single level.

In the previous work [3], we have already assessed some of the implications of NVM, considering elements as computing time and power consumption. This work extends it, presenting an evaluation of a search engine application in a hypothetical computer with persistent main memory. Through the use of simulation, we aim to identify the major system-level impacts of persistent main memory in latency and energy. To the best of our knowledge, this is the first study evaluating both PCRAM and Memristor using a programming interface specific to persistent main memory, while considering timing, energy, and impacts on code development effort and complexity.

2. Emerging memory technologies

There are several new non-volatile memory (NVM) technologies under research today [4]. This study focuses on two of these technologies: phase-change RAM (PCRAM) and Memristor, since they are among the most mature candidate technologies for DRAM replacement. Table 1 compares the main properties of traditional memory/storage technologies with PCRAM and Memristor. Data was obtained from [2,5,4,6–9].

2.1. Phase-change RAM (PCRAM)

Phase-change random access memory (also called PCRAM, PRAM or PCM) is currently the most mature of the new memory technologies under research. It relies on phase-change materials that exist in two different phases with distinct properties: an amorphous phase, characterized by high electrical resistivity, and a crystalline phase, characterized by low electrical resistivity [10]. These two phases can be repeatedly and rapidly cycled by applying heat to the material [10,2].

[*] Corresponding author. Tel.: +55 51 21213915.
E-mail address: taciano.perez@hp.com (T. Perez).

0026-2692/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.mejo.2013.11.001

Table 1
Comparison of memory/storage technologies.

Maturity	**SRAM** Product	**DRAM** Product	**Disk** Product	**NAND Flash** Product	**PCRAM** Adv. Dev.	**Memristor** Early dev.
Read latency	< 10 ns	10–60 ns	8.5 ms	25 μs	48 ns	10–100 ns
Write latency	< 10 ns	10–60 ns	9.5 ms	200 μs	40–150 ns	10–100 ns
Static power	Yes	Yes	Yes	No	No	No
Endurance	$> 10^{15}$	$> 10^{15}$	$> 10^{15}$	$> 10^{4}$	$> 10^{8}$	$> 10^{12}$
Nonvolatility	No	No	Yes	Yes	Yes	Yes

2.2. Memristor

A Memristor is a two-terminal device whose resistance depends on the magnitude and polarity of the voltage applied to it and the length of time that voltage has been applied. When the voltage is turned off, the Memristor remembers its most recent resistance until the next time it is turned on. The property of retaining resistance values means that a Memristor can be used as a nonvolatile memory [11].

This first Memristor device created consisted of a crossbar of platinum wires with titanium dioxide (TiO_2) switches. Each switch consists of a lower layer of stoichiometric titanium dioxide (TiO_2), which is electrically insulating, and an upper layer of oxygen-deficient titanium dioxide (TiO_{2-x}), which is conductive. The size of each layer can be changed by applying voltage to the top electrode. If a positive voltage is applied, the TiO_{2-x} layer thickness increases and the switch becomes conductive (ON state). A negative voltage has the opposite effect (OFF state) [12,13,11]. Several oxides other than TiO_2 are known to present similar bipolar resistive switching, and there are multiple research projects in motion to explore these other materials for similar memory device implementations [2].

3. Persistent main memory

These novel non-volatile memory technologies can potentially make a significant amount of persistent main memory available to computer systems. It would allow a collapse of two different levels of the storage hierarchy – main memory and persistent storage – into a single level, something that has never been practically feasible before. The advent of main memory as the primary persistent storage may deeply affect most computing layers, including application software, operating system, busses, memory system and their interaction with other devices, such as processors and I/O adapters [14,15]. In order to fully assess system-wide impacts on latency, energy, heat, space and cost, it is required to take into account all these different layers when modeling or simulating a hypothetical computer system with persistent main memory.

Initial proposals for the application of NVM technologies evaluated the individual replacement of existing memory hierarchy levels (such as processor cache, main memory and persistent storage) by NVM counterparts, with gains of performance and efficiency at subsystem level [16–19]. Most of these proposals do not imply a radical redesign of computing systems as a whole, but localized changes to specific subsystems.

More recently, proposals for more radical system redesigns were published. A good example is the architecture of Nanostores [15] that proposes parallel systems with a massive number of low-cost processors co-located with non-volatile data stores. This system is targeted for data-centric workloads, such as search, sort and video transcoding, and particularly suited for scale-out server environments.

Table 2
Experimental target configuration setup (guest).

Processor	x86-64 (Hammer) 20 MHz (single-core)
L1 Cache	Size: 16 Kb (D-cache)+16 Kb (I-cache) Associativity: 4-way (D-cache), 2-way (I-cache) Penalty: 100–1000 ps Replacement policy: LRU
L2 Cache	Size: 512 Kb Associativity: 8 × Penalty: 10 ns Replacement policy: LRU
Main memory	Size: 4096 MB Penalty: technology-dependent (see Table 3)
Disk	20 GB
OS	Fedora Core release 5 (Bordeaux) Kernel: 2.6.15-1.2054_FC5

In the present study, we explore a simple commodity system where DRAM is fully replaced by non-volatile memory, either Memristor or PCRAM. No other significant changes will be applied. The system aspects being analyzed are timing and energy.

4. Experimental setup

4.1. Simulation environment

The simulated system (guest) is a single-processor x86 64-bit (Hammer) computer with 4 GB of main memory. It was simulated using Virtutech Simics [20], a full-system simulator. The main parameters of the simulation are described in Table 2. This setup enables us to exercise variations in the memory technology parameters in a very simple commodity system. We believe that future systems with persistent main memory will have different characteristics, such as a much larger memory size (comparable to current hard disks), and different physical memory organization, since JEDEC's DDRx does not support hundreds of Gigabytes. Our purpose is to have a first-order approximation of the impact of persistent main memory in current designs.

We executed a set of computing tasks (described in the next section) in three different scenarios: DRAM, PCRAM and Memristor. For each scenario, the memory latency and energy parameters were set as shown in Table 3, using a customized version of the trans-staller module in Simics. The latency values at device-level were derived from [21–23]. A low clock frequency was used in order to expedite test execution. As validation, we observed the ratio between in-memory/file-based using DRAM in both the simulated computer and real ones, and found them to be consistent. We believe that the number of memory loads/stores,

Table 3
Technology parameters for the three simulated scenarios: DRAM, Memristor and PCRAM.

Parameter	DRAM	Memristor	PCRAM
Read latency (ns)	50	100	50
Write latency (ns)	48	100	150
Read energy (nJ)	3.4539	0.129	6.75
Write energy (nJ)	3.4475	1.109	9.872
Refresh power (mW)	0.0867	0	0
Feature size (nm)	45	45	45

which is the other main metric gathered during the simulation, is not significantly affected by higher process frequencies.

In order to estimate the overall energy consumption of each memory technology scenarios, we used an energy model that considers two separate elements:

1. *Dynamic energy* – the energy consumed in order to read/write memory addresses.
2. *Refresh energy* – the energy consumed to keep the memory contents alive. It is only relevant for DRAM, since PCRAM and Memristor do not need refresh due to their own persistent nature.

Subthreshold power leakage is the third important component of the total energy consumption. NVM should have leakage at least similar to DRAM, or possibly even lower, since idle memory banks can be turned off. This study did not consider leakage due to the lack of published information on the leakage of Memristor memory devices at this moment.

The energy parameters in Table 3 were obtained using CACTI [24] (for DRAM) and NVSim [25] (for PCRAM and Memristor). Latency and energy parameters refer to memory operations (load/store).

The Virtutech Simics architectural simulator (host) was executed in a server with 32 Xeon 2.4 GHz cores, 64 GB of memory and 115 GB of local storage.

The present study does not address mechanisms to improve NVM endurance, assuming that wear leveling techniques such as those described in [21,22,8,16] will be employed.

The next section describes the computing tasks used as workload for our experiments.

4.2. Workload

As workload we have chosen the tasks of indexing and searching terms in a set of text documents. This is a critical, well-known real-world task, performed both in personal computers (e.g. e-mail search) and huge parallel machines (e.g. web search), which depends critically on persistent data that can be easily held either in memory or in files [26].

We have implemented and executed this task using two different approaches: (1) a traditional implementation using files; (2) an implementation using a programming interface specific for persistent main memory. The motivation is to investigate if alternative programming interfaces can have significant impact on the overall performance.

As interface for memory persistence, we used Mnemosyne [14], a modern framework available as open-source. Mnemosyne enables programmers to make in-memory data structures persistent without converting it to serialized formats, bypassing software layers such as system calls, file systems and device drivers, using transactional memory constructs in order to ensure consistency. It provides a simple interface for the programmer, ensuring consistency across modifications and being compatible with existing commodity processors.

In order to make a data structure persistent in Mnemosyne, it has either to be declared using a modifier specific to indicate persistence, or alocated with a persistent memory allocation function (pmalloc). During every program execution, persistent data structures keep their previously set values.

We have created a basic search engine using concepts presented in [26]. The features of our search engine implementation can be thus summarized:

- Generates inverted index, receiving plain text files as input.
- Allows boolean retrieval of documents by term.
- Uses Blocked Sort-Based Indexing (BSBI) for the file-based implementation approach.
- Uses a Red-Black Tree data structure for the Mnemosyne implementation approach.

In order to keep it simple, the search engine does not implement more sophisticated features such as index compression, document scoring or stemming. It is organized around three main subsystems:

1. *Index and search algorithms* – the common algorithms used for indexing and searching terms independent of the implementation approach. It includes (a) reading input files (documents), assigning a docID for each one, tokenizing them and issuing term/termID pairs; (b) intersecting the results in case of multiple term searches; (c) a "main" function to parse input parameters and invoke index or search actions.
2. *File-based storage* – implementation of storage in the file system using Blocked Sort-Based Indexing (BSBI), a concept described with details in [26]. For each document and each term unique numeric IDs are assigned. The index stores in a file a pair termID/docID for every document that contains a given term. A set of files containing such pairs compose the BSB Index. The contents of the files are always sorted by termID, in order to make the queries more efficient (only the files containing the desired terms are scanned when a query is issued).
3. *In-memory storage* – implementation of in-memory storage using a Red-Black Tree persisted through Mnemosyne. It uses a modified version of an open-source Red-Black Tree [27], changed to allocate internal data structures using Mnemosyne.

The BSBI indexing process requires reading each input file and creating intermediate output index files with pairs termID/docID (BSBI entries), without repeating entries. After all the intermediate index files are created, they are merged, i.e., a new and final set of index files with the ordering of all BSBI entries. Index files have a maximum size, and when they fill up a new file is created. The goal is to avoid keeping all the indices in memory. In our implementation, each BSBI entry is 8 bytes long, and the maximum BSBI index file size is 256 KB. There are three indices in memory: (a) a term/termID map; (b) a doc/docID map; and (c) an index to the first and last termID contained by each index file.

Table 4
Workload development: size and effort metrics.

#	Scope	LOC	# Classes	Dev. hours (h)
1	**Index & search**	603	7	30
2	**File system**	499	3	40
3	**In-memory**	241	3	8
4	**Red-Black Tree**	440	N/A	4
5	**Total**	1783	13	77

These indices are also backed up to files, and loaded to memory during program startup. The search process receives a set of terms. Each one is translated to its corresponding termID, then the index in memory is consulted to identify which file(s) contain(s) the BSBI entries for that termID. Finally, the file is read sequentially until these references are found, and its corresponding docIDs are identified. This process is repeated for every search term, and at the end they are intersected to return the results of a logical AND operation between the terms.

Table 4 displays the main metrics related with development size, complexity and effort. These metrics were chosen to allow a comparison of the two different implementation approaches in terms of development efficiency. The number of lines of code and comments help understanding the software size. The number of classes associated with the number of lines of code can be used as an indicator of complexity. The number of development hours informs the effort applied to develop each scenario. Line 1 includes components common to both implementation approaches. Line 2 includes components specific to the file-based approach. Lines 3 and 4 include components specific to the Mnemosyne approach. Line 5 lists the total metrics for both approaches.

4.3. Corpus

A set of textual data was used as the corpus to be indexed and searched. It consisted of Shakespeare complete works, obtained as a set of plain text files from the University of Sydney [28]. It contains a set of 44 documents with total size of 7372 KB containing 24,427 distinct terms.

The in-memory storage scenario assumes that persistent memory is the main storage location, and its contents will not be swapped to secondary storage. For this reason, a corpus with a total size larger than the processor caches but smaller than memory is considered sufficient for the experiment.

5. Results and discussion

We have measured individually the execution of three different tasks:

1. *Index generation* – the task of reading the complete corpus and generating an inverted index, either in memory or disk.
2. *Search 1 (simple)* – the task of searching the index for a given term ("Brutus").
3. *Search 2 (intersection)* – the task of searching the index for two given terms ("Brutus" and "Calpurnia") and intersecting the results (logical "AND").

All three tasks (index, search 1 and search 2) were executed using both implementation approaches (file-based, Mnemosyne) in each of the three simulated memory technologies (DRAM, Memristor and PCRAM). Every execution was repeated three times. The running time for each execution ranged between a few minutes and around 3 h.

5.1. Implementation approach

A major benefit of using memory persistence (Mnemosyne) is relative to application development and complexity. Table 4 shows that development using memory persistence produces less than half lines of code and is more than 4 × faster to develop than using disk.

These results support the case for not only replacing current hardware designs using NVM, but also taking different approaches at software level, such as memory persistence.

Table 5
Execution times for experimental workloads, in seconds.

Approach	Task	DRAM	Memristor	PCRAM
File-based	Index	550.83	771.22	726.44
File-based	Search 1	13.24	20.65	13.26
File-based	Search 2	13.36	19.75	13.34
Mnemosyne	Index	–	354.49	84.38
Mnemosyne	Search 1	–	0.41	0.39
Mnemosyne	Search 2	–	0.43	0.40

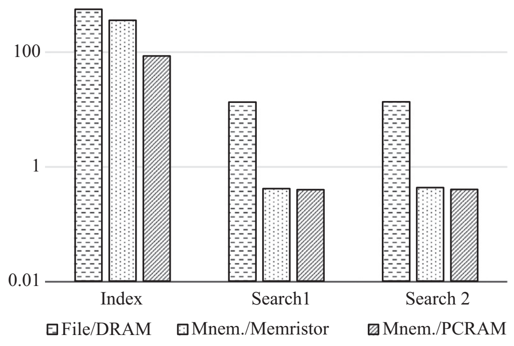

Fig. 1. Comparison of File/DRAM vs. Mnemosyne/Memristor and Mnemosyne/PCRAM task execution times (in s), using a logarithmic scale.

5.2. Execution time

Complete execution times are shown in Table 5. Fig. 1 highlights a comparison between the file-based approach/DRAM and Mnemosyne using both Memristor and PCRAM. When using the traditional file-based approach, Memristor and PCRAM executions are up to 56% slower than DRAM. This is explained by the larger latency to read/write Memristor and write PCRAM, as previously shown in Table 3. PCRAM latency was considered equivalent to DRAM, and thus tasks dominated by reads (i.e., searches) in PCRAM have results similar to DRAM.

Although Memristor is considered here 2 × slower, and PCRAM write latency 3 × slower than DRAM, the average overall task performance impact is less steep. This is consistent with the results published in similar studies [8,16,21,22,29]. The main factor behind this phenomenon is the high rate of L1 and L2 cache hits in a typical workload, which in our experiments were consistently above 93% for L1 Read and 92% for L2 Read. This observation reinforces the idea that main memory using Memristor or PCRAM still needs processor caches in order to avoid severe performance penalties. Design proposals for caches using NVM are explored in [5,17–19,30].

On the other hand, the implementation approach using memory persistence (Mnemosyne) is significantly faster than its counterpart using file-based persistence in secondary storage. Indexing the input text files using memory persistence is 1.5 × faster using Memristor and 7 × faster using PCRAM. Search time is in average more than 33 × faster than its file-based counterpart.

The file-based approach using secondary storage is slower because it must go through the I/O subsystem to perform durable reads and writes. Operating systems typically use memory caches for most I/O operations reducing the latency impact, but it is still a significantly longer and more complex affair than simply issuing load/store memory instructions. In our experiment, the index operation using file-based persistence reads 7.9 MB from secondary storage, and writes 2.5 MB to it; the search operation reads in average 2.3 MB from secondary storage, and writes 5 KB to it. The in-memory approach, as the name implies, does not require I/O to secondary storage.

Table 6
Energy consumption of experimental workloads, in J.

Technology	Approach	Task	Dynamic	Refresh	Total
DRAM	File-based	Index	14.782	47.758	62.539
		Search 1	0.818	1.148	1.966
		Search 2	0.822	1.159	1.981
Memristor	File-based	Index	2.344	–	2.344
		Search 1	0.018	–	0.018
		Search 2	0.121	–	0.121
	Mnemosyne	Index	2.187	–	2.187
		Search 1	0.007	–	0.007
		Search 2	0.009	–	0.009
PCRAM	File-based	Index	36.226	–	36.226
		Search 1	1.916	–	1.916
		Search 2	1.878	–	1.878
	Mnemosyne	Index	34.670	–	34.670
		Search 1	0.121	–	0.121
		Search 2	0.130	–	0.130

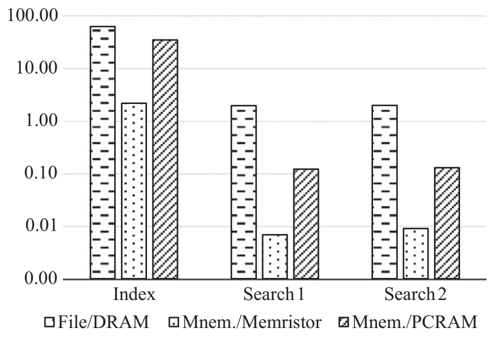

Fig. 2. Comparison of File/DRAM vs. Mnemosyne/Memristor and Mnemosyne/PCRAM total energy consumption (in J), using a logarithmic scale.

5.3. *Energy*

The detailed energy consumption results for each execution can be seen in Table 6. Fig. 2 highlights a comparison between the file-based approach/DRAM and Mnemosyne using both Memristor and PCRAM. The main impact of non-volatile memories (PCRAM and Mnemosyne) is on the energy footprint of main memory, which is significantly smaller in non-volatile technologies than DRAM, due to the absence of energy consumption for refresh. Using Memristor with memory persistence (Mnemosyne) consumes 33 × less memory energy than using DRAM/Disk with a file-based approach.

The main memory energy footprint is similar for the file-based and in-memory approaches when using persistent main memory. However, the overall energy consumption becomes significant when we consider the energy used for secondary storage (HDD/SDD), which is not modeled in this work.

When comparing Memristor and PCRAM as base technology for persistent main memory, we observe that PCRAM has a slight advantage regarding execution speed, given the latency parameters used in this study. However, Memristor is significantly superior from the energy standpoint, with savings up to 15 × less consumption.

6. Related research

Other studies have evaluated the usage of emerging non-volatile memory technologies as persistent main memory using appropriate programming interfaces.

Mnemosyne [14] is a simple interface for programming with persistent memory, allowing creation and management of such memory and ensuring consistency in the presence of failures. Their study considered OpenLDAP and TokyoCabinet (a key-value store) as performance benchmarks and modeled PCRAM memory. The current study uses Mnemosyne as persistent memory programming interface, adding Memristor modeling and considering energy impacts, which were not part of the original Mnemosyne study.

NV-Heaps [31] is another framework for exploiting persistent memory, allowing the declaration of non-volatile heaps containing persistent objects. PCRAM and STT-MRAM technologies were modeled using workloads that manipulated several data structures such as trees and hash tables for benchmarking performance. The current study also models Memristor and considers energy impacts.

To the best of our knowledge, the current study is the first that evaluates the usage of PCRAM and Memristor using a memory persistence programming interface and considering both timing and energy aspects, as well as evaluating development effort and complexity concerns.

7. Conclusion

This work presented an evaluation of a search engine application running in a hypothetical computer with persistent main memory, through the use of experimental models and simulations, aiming to identify the major system-level impacts of persistent main memory in latency and energy.

Our observations confirm the previous observations that main memory using Memristor or PCRAM still needs processor caches in order to avoid performance penalties. Overall performance of non-volatile technologies such as main memory using memory persistence programming interfaces (Mnemosyne) is considerably superior to traditional file-based implementations on top of DRAM and hard disks.

The energy footprint of non-volatile memory technologies as main memory is significantly smaller than DRAM. In our experiments, Memristor with memory persistence consumes 33 × less memory energy than using DRAM/Disk with a file-based approach.

One remaining challenge for using these technologies as main memory is their low endurance. Wear leveling techniques such as those described in [21,22,8,16] are expected to contribute positively.

The experimental results support the feasibility of employing emerging non-volatile memory technologies such as persistent main memory. Our study indicates that performance penalties should be mild and energy improvements should be significant.

This study compared the development and execution of a search engine application using both a traditional filesystem approach and a memory persistence approach (Mnemosyne). Current programming interfaces have separate abstractions for handling memory (data structures) and secondary storage (files, databases), and a good deal of development and processing effort is directed towards moving data between these two layers. Development using in-memory persistence produced less than half lines of code and was more than 4 × faster to develop than using a file-based approach. These results support the case for different approaches at software level, such as memory persistence. However, in order to successfully replace traditional filesystem or database approaches, it is necessary to have memory persistence frameworks that allow sharing data seamlessly between different programs and languages, as filesystems and databases do. This is a research problem that is still pending.

We conclude that in order to reap the major rewards potentially offered by persistent main memory, it is worthwhile to take new programming approaches that do not conceptually distinguish main memory from secondary storage.

References

[1] B. Jacob, S. Ng, D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann Pub, 2007.

[2] G. Burr, B. Kurdi, J. Scott, C. Lam, K. Gopalakrishnan, R. Shenoy, Overview of candidate device technologies for storage-class memory, IBM J. Res. Dev. 52 (4) (2008) 449–464.

[3] T. Perez, N. Calazans, C. De Rose, A preliminary study on system-level impact of persistent main memory, in: 13rd International Symposium on Quality Electronic Design (ISQED), 2012, pp. 85–90.

[4] M. Kryder, C. Kim, After hard drives – what comes next? IEEE Trans. Magn. 45 (10) (2009) 3406–3413.

[5] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, Y. Chen, Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement, in: 45th ACM/IEEE Design Automation Conference (DAC), 2008, pp. 554–559.

[6] D. Lewis, H. Lee, Architectural evaluation of 3D stacked RRAM caches, in: IEEE International Conference on 3D System Integration (3DIC), 2009, pp. 1–4.

[7] J. Mogul, E. Argollo, M. Shah, P. Faraboschi, Operating system support for NVM+ DRAM hybrid main memory, in: 12th Conference on Hot Topics in Operating Systems, USENIX Association, 2009, p. 1–5.

[8] M. Qureshi, V. Srinivasan, J. Rivers, Scalable high performance main memory system using phase-change memory technology, in: 36th Annual International Symposium on Computer Architecture (ISCA), ACM, 2009, pp. 24–33.

[9] M.-J. Lee, C.B. Lee, D. Lee, S.R. Lee, M. Chang, J.H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, et al., A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures, Nat. Mater. 10 (8) (2011) 625–630.

[10] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S. Chen, H. Lung, et al., Phase-change random access memory: a scalable technology, IBM J. Res. Dev. 52 (4.5) (2010) 465–479.

[11] R. Williams, How we found the missing memristor, IEEE Spectr. 45 (12) (2008) 28–35.

[12] D. Strukov, G. Snider, D. Stewart, R. Williams, The missing memristor found, Nature 453 (7191) (2008) 80–83.

[13] J. Yang, M. Pickett, X. Li, D. Ohlberg, D. Stewart, R. Williams, Memristive switching mechanism for metal/oxide/metal nanodevices, Nat. Nanotechnol. 3 (7) (2008) 429–433.

[14] H. Volos, A. Tack, M. Swift, Mnemosyne: Lightweight persistent memory, in: Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 91–104.

[15] D. Roberts, J. Chang, P. Ranganathan, T. Mudge, Is Storage Hierarchy Dead? Co-located Compute-Storage NVRAM-based Architectures for Data-Centric Workloads, Technical Report, HP Labs, 2010.

[16] P. Zhou, B. Zhao, J. Yang, Y. Zhang, A durable and energy efficient main memory using phase change memory technology, in: 36th Annual International Symposium on Computer Architecture (ISCA), 2009, pp. 14–23.

[17] C. Koh, W. Wong, Y. Chen, H. Li, The Salvage Cache: A fault-tolerant cache architecture for next-generation memory technologies, in: IEEE International Conference on Computer Design (ICCD), 2009, pp. 268–274.

[18] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, Y. Xie, Hybrid cache architecture with disparate memory technologies, in: 36th Annual International Symposium on Computer Architecture (ISCA), ACM, New York, NY, USA, 2009, pp. 34–45.

[19] X. Wu, J. Li, L. Zhang, E. Speight, Y. Xie, Power and performance of read-write aware hybrid caches with non-volatile memories, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2009, pp. 737–742.

[20] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Haallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: a full system simulation platform, IEEE Comput. 35 (2) (2002) 50–58.

[21] B. Lee, E. Ipek, O. Mutlu, D. Burger, Architecting phase change memory as a scalable dram alternative, ACM SIGARCH Comput. Archit. News 37 (3) (2009) 2–13.

[22] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, D. Burger, Phase-change technology and the future of main memory, IEEE Micro 30 (1) (2010) 143.

[23] P. Ranganathan, From microprocessors to nanostores: rethinking data-centric systems, IEEE Comput. 44 (1) (2011) 39–48.

[24] S. Thoziyoor, N. Muralimanohar, J. Ahn, N. Jouppi, CACTI 5.1, HP Laboratories, April 2008.

[25] X. Dong, C. Xu, Y. Xie, N.P. Jouppi, Nvsim: a circuit-level performance, energy, and area model for emerging nonvolatile memory, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 31 (7) (2012) 994–1007.

[26] C. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.

[27] E. Martinian, Open Source Red-Black Tree ⟨http://web.mit.edu/~emin/www.old/source_code/red_black_tree/index.html⟩, 2013 (accessed 2013-04-09).

[28] University of Sidney, Shakespeare Corpus ⟨http://sydney.edu.au/engineering/it/~matty/Shakespeare/⟩, 2013 (accessed: 2013-04-09).

[29] G. Dhiman, R. Ayoub, T. Rosing, PDRAM: A hybrid PRAM and DRAM main memory system, in: 46th ACM/IEEE Design Automation Conference (DAC), IEEE, 2009, pp. 664–669.

[30] S. Sardashti, D.A. Wood, Unifi: leveraging non-volatile memories for a unified fault tolerance and idle power management technique, in: 26th ACM International Conference on Supercomputing (ICS), 2012, pp. 59–68.

[31] J. Coburn, A. Caulfield, A. Akel, L. Grupp, R. Gupta, R. Jhala, S. Swanson, Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories, in: Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 105–118.