Invited Paper

FPGA-based Prototyping Systems for Emerging Memory Technologies

Taemin Lee, Dongki Kim, Hyunsun Park, Sungjoo Yoo, Sunggu Lee
Embedded System Architecture Laboratory
Department of Electrical Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, Korea
{ltmin, dongki.kim, lena0911, sungjoo.yoo, slee}@postech.ac.kr

***Abstract*— As DRAM faces scaling limit, several new memory technologies are considered as candidates for replacing or complementing DRAM main memory. Compared to DRAM, the new memories have two major differences, non-volatility and write overhead in terms of endurance, latency and power. We built two different FPGA-based evaluation boards to evaluate hardware and software designs for new-memory based main memory; one with a DRAM subsystem having parameterizable latency and non-volatility emulation, and the other with the real chips of new memory namely phase-change RAM (PRAM). We experimented primitive functions and SQLite-based benchmarks on Linux, verifying the workings of new functionalities, e.g., non-volatility and evaluating the impacts of new memory on software performance. In our experiments, we also demonstrated the impact of new memory-aware software/hardware designs on program performance on a DRAM/PRAM hybrid memory.**

Keywords—DRAM replacement; main memory; SQLite benchmark

I. INTRODUCTION

For decades, dynamic random access memory (DRAM) has been an irreplaceable technology in main memory due to low cost. However, DRAM is expected to suffer from scaling limits in sub-20nm [1], increasing demands for non-charge based novel memory technologies i.e., phase-change RAM (PRAM) [2], spin-transfer torque RAM (STT-RAM) [3], and resistive RAM (ReRAM) [4]. Especially, large-capacity PRAM chips are already available [5].

As new memory technologies have common interesting characteristics such as byte-addressability and non-volatility, these have been strong candidates for replacing or complementing both main memory and storage. For main memory, new memories have stronger points on non-volatility and semiconductor scaling than DRAM. For storage, new memories are byte-addressable and faster than existing technologies, HDD and flash memory. In this paper, we focus on utilizing new memory as main memory.

New memory technology also has limitations compared to DRAM. First, write operation has huge overhead in terms of latency, power and endurance. For instance, PRAM has long write latency of 40~490us for one page write [6][7][8]. Under a peak write power constraint, high write power consumption limits the number of PRAM cells to be programmed concurrently, i.e., write bandwidth. The write endurance of RPAM is known to be only 10^8 per cell, compared with 10^{16} in DRAM. In order to compensate for weak points and exploit the benefits of new memories, recently, there have been many studies in hardware architecture and software [8][9][10]. Most of existing works rely on simulations and estimations due to lack of real hardware prototype. Because full system simulations are not practically available due to very long simulation runtime, partial system simulations, e.g., 1 billion instruction simulation which corresponds to less than 1 second in real time in most systems, are widely used. Such a limitation of simulation-based approach motivated us to develop real prototype systems.

We developed two evaluation boards using Xilinx Zynq ARM-FPGA SoC. The first board is called a non-volatility emulation board. On the DRAM-based board, we implemented two functions; parameterizable read/write latency to main memory and non-volatility emulation. The board helps evaluate the performance impact of new memory and verify non-volatility-aware software optimizations. The other is equipped with a real PRAM DIMM board with LPDDR2 interface, called PRAM-based board. We built a PRAM-based DIMM daughterboard and an in-house PRAM controller. With the PRAM board, we evaluate the performance of real PRAM-based system.

Both new memories (DRAM with new memory emulation and real PRAM) are attached to the memory bus. Operating system (OS), i.e., Linux in our case runs on the ARM processor in the FPGA accessing the new memory as a cacheable region. In addition, in the case of PRAM-based board, we also have an additional DRAM on the memory bus. Thus, both memories can be directly compared by running the same programs on each of two memories.

Our contribution is summarized as follows.

- Real prototype systems. We developed full functional real prototype systems to help evaluate the performance of operating system and application software running on the ARM processor and new memory-based main memory.

978-1-4799-6851-0/14/$31.00

- Non-volatility emulation board. It enables us to emulate the performance of new memories by adjusting memory access latency. Non-volatility is also emulated on a memory region basis.
- Real PRAM-based board. It can evaluate/analyze the impact of PRAM on the performance of main memory operations. We also provide an optimization of PRAM controller with a duplicated program buffer (called virtual page) to boost PRAM write performance, triggering a buffered write mode.

This paper is organized as follows, Section II gives related work. Section III introduces the details of our FPGA-based DRAM board, called non-volatility emulation board. Section IV describes PRAM-based. Section V summarizes the paper.

II. Related Work

Recently, there have been active research on new memory-based main memory architectures and software design methods. Hybrid DRAM/new memory is one of popular architectures. Qureshi, et al. propose utilizing a small DRAM buffer, in front of large PRAM in order to exploit the benefits of low latency of DRAM and high capacity of PRAM [11]. H. Park, et al. apply the concept of cache decay to evict cold data from DRAM thereby reducing the overhead of DRAM refresh in the hybrid DRAM/PRAM main memory [12]. Lee, et al. show the feasibility of PRAM-only main memory [17]. STT-RAM tends to replace DRAM main memory by itself. Jin, et al. present an optimization of STT-RAM layout with faster latency than DRAM while having comparable cell size [13]. Kultursay, et al. show that the energy efficiency of STT-RAM main memory is 60% better than that of DRAM [3]. Software-based studies mostly target non-volatility. Coburn, et al. present lightweight and high-performance persistent objects called NV-heaps on the non-volatile main memory [10]. Lee et al. integrate page cache and journal buffer by exploiting the non-volatility of main memory [14].

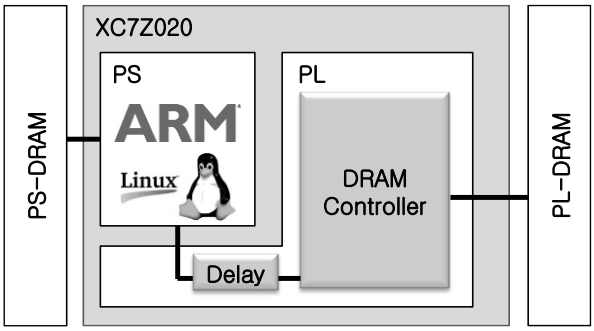
There were several prototype systems for new memories. Akel, et al. present a prototype PRAM-based SSD called Onyx [15]. Onyx uses SPI interface PRAM and has total capacity of 8 GB. Caulfield, et al. present a high-performance storage array architecture, called Moneta, using multiple FPGAs and 64GB DRAM [16]. Moneta shows that fast and byte-addressable storage can significantly reduce storage latency. These works utilize new memory (emulation in [16]) for storage purposes while ours target main memory applications.

III. Non-Volatility Emulation System

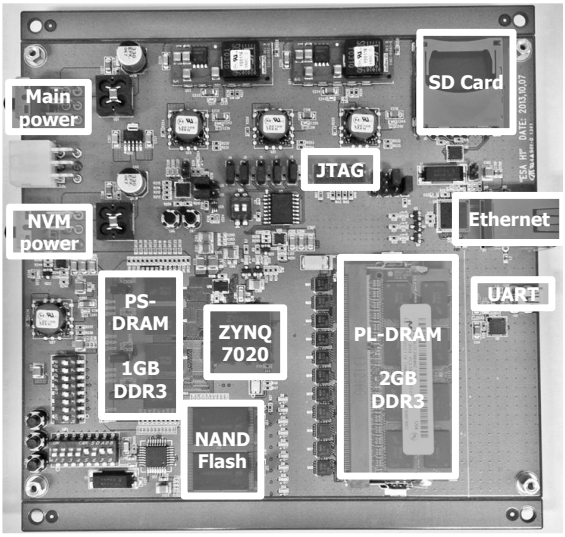
Fig. 1 shows our non-volatility emulation board with Xilinx Zynq XC7Z020. Zynq consists of ARM processor subsystem called processing system (PS) and FPGA programmable logic (PL) on a single chip. PS is based on ARM Cortex-A9 processor with L2 caches. PL has 85K logic cells (Xilinx Artix-7). PS and PL are connected by AXI interface. We can run OSs on PS using SD card; Linux, U-BOOT, and a standalone OS provided by Xilinx. In addition, JTAG, UART, Ethernet, DIP switches, and LEDs are available.

A. Memory Subsystem

Fig. 1 (b) shows that the board has two DRAM subsystems. PS has a hard-wired DRAM subsystem using PS internal DDR controller, called PS-DRAM. As PS logic is not programmable, we implement an additional DRAM subsystem on PL-side, called PL-DRAM. PL-DRAM controller is implemented using Xilinx 7-series MIG IP, and we added an interface logic of parameterizable latency between the memory controller and AXI bus.



(a) Block diagram



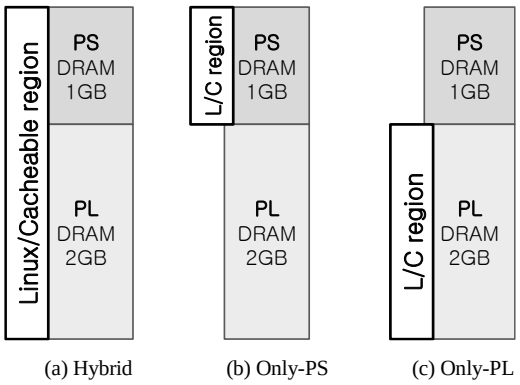
(b) Top view

Fig. 1. Non-volatility emulation board

Table I compares PS-DRAM and PL-DRAM. PS-DRAM is clocked with 533MHz (DDR3-1066) while PL-DRAM operates speed of 400MHz (DDR3-800) which is the maximum frequency of Xilinx memory controller (MIG IP). PL-DRAM has additional hardware layer, AXI bus which gives additional latency. PL-DRAM has separate power switch named non-volatile memory (NVM) power for non-volatility emulation.

TABLE I. Comparison between PS-DRAM and PL-DRAM

Component	PS-DRAM	PL-DRAM
Frequency	533MHz (DDR3-1066)	400MHz (DDR3-800)
Data width	8bx4=32b (DIMM)	8bx8=64b (So-DIMM)
Data path	DDR3	DDR3 – AXI
Power source	Main power (merged)	NVM power (separate)



(a) Hybrid (b) Only-PS (c) Only-PL

Fig. 2. Configurations of Linux/cacheable region on physical address space

Note that PL-DRAM is slower than PS-DRAM as shown in Table 1. In our measurement, the latency of random access is ~120ns in PS-DRAM while PL-DRAM takes ~310ns.

Our board has 1GB PS-DRAM and 2GB PL-DRAM, total 3GB main memory. Early 1GB (0 ~ 0x3FFFFFFF) is allocated to PS-DRAM, and next 2GB (0x40000000 ~ 0xbFFFFFFF) is belong to PL-DRAM. Linux/cacheable (L/C) region can be located on a designer's choice. Fig. 2 exemplifies three configurations. We modify device tree blob of Linux to determine L/C region. Given an L/C region, the remaining address region of physical memory can be accessed as a non-cacheable device memory via mmap(), i.e., 'mmap /dev/mem'.

B. Non-Volatility Emulation Function

Non-volatility is emulated with PL-DRAM. DRAM needs refresh to maintain its data, typically with a period of 64ms. To make PL-DRAM non-volatile, we implement a NVM power switch for PL-DRAM. Once main power is down, the system loses operating power except PL-DRAM. In this case, PL-DRAM automatically enters self-refresh state and keeps the stored data. As a result PL-DRAM retains previous data after the system reboot emulating non-volatility. The non-volatility can be set on any program region which is determined by the user.

C. Performance Emulation Function

New memories are expected to have lower performance than DRAM. Especially write performance is much lower than read performance. We implemented individual delay modules for each read/write data path of PL-side DRAM. To be specific, we delay handshake signals (ARREADY) on AW and AR channels of PL-DRAM's AXI slave interface. Therefore, we can independently control the read and write latency of PL-DRAM.

D. Experiments

We experimented three cases: performance test of cacheable region, non-volatility emulation, and performance emulation test.

We configure the main memory address space to experiment five different memory regions; non-cacheable PS-DRAM, cacheable PS-DRAM, non-cacheable PL-DRAM, cacheable PL-DRAM, and hybrid memory. Fig. 3 shows the latency measurement of write requests to each memory region. We use mmap() to allocate each memory regions except for hybrid memory. Hybrid memory can be allocated using malloc() since this region is managed by OS.

We generate write traffics with a total footprint size of 64MB to make sure that sufficient cache conflicts occur. Pattern DIST=64 generates write requests to addresses of multiples of 64 which incurs frequent misses in the L1 and L2 caches. Pattern DIST=1 generates sequential write requests, makes best use of cache within the same footprint size.

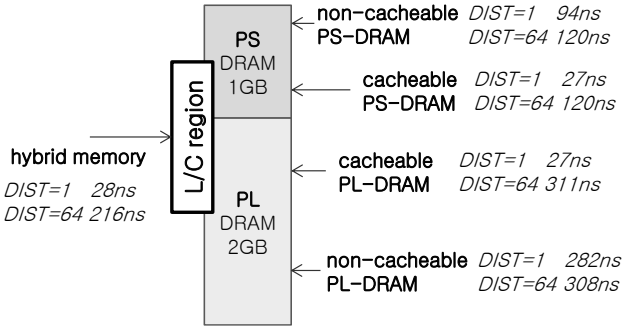


Fig. 3. Average latency for each memory region

Fig. 3 shows that PL-DRAM, due to the additional latency of its AXI bus layer, gives much (~190ns) longer average latency when frequent cache misses occur. The figure also shows that caches help reduce average latency since pattern DIST=1 gives a low latency under 30ns in L/C region.

Fig. 4 shows the non-volatility functional experiment. We use U-BOOT, which can access directly to specific physical address. Our experiment scenario is followed

- STEP1: Write arbitrary data on PL-DRAM

  (U-BOOT command: mw.b 80000000 08 100)
- Turn main power off. Wait for a few seconds. Then, turn main power on.
- STEP2: Check to see if the previous data still remain.

  (U-BOOT command: md 80000000)
- STEP3: Repeat with PL-DRAM power off.

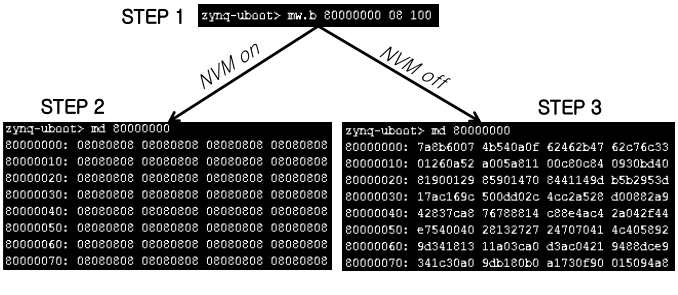
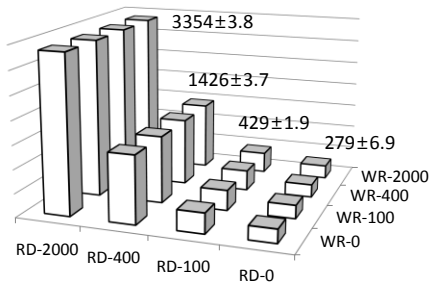


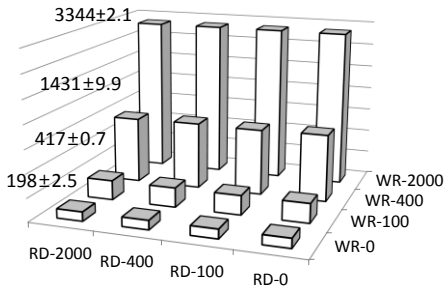
Fig. 4. Non-volatility experiment

Although main power is turned off, the NVM power provides PL-DRAM to perform self-refresh. STEP2 shows that data is maintained under system shutdown. In STEP3 which

also turns off the separate power of PL-DRAM, random data are observed like a typical volatile DRAM as PL-DRAM loses its data.



(a) Average latency of read



(b) Average latency of write

Fig. 5. Average read/write latency [ns] vs. additional delay [cycle]

We evaluated the performance of primitive functions in terms of latency and bandwidth for new memory emulation. Fig. 5 shows average read/write latency of PL-DRAM measured while varying additional latency. For instance, WR-400 inserts 400 additional cycles to emulate slow new memory write operations. As we insert larger additional cycles on delay module, average latency also increases. Also, we can control the read/write latency independently.

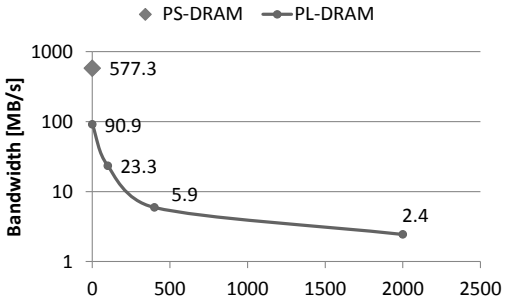


Fig. 6. Memset bandwidth [MB/s] vs. additional delay [cycle].

Fig. 6 shows the bandwidth of 4kB memset function. PS-DRAM gives a sequential write bandwidth of 577.3MB/s, which is much higher than that of PL-DRAM, 90.9MB/s, which is due to the lower clock frequency and additional AXI layer in PL-DRAM subsystem. The bandwidth monotonically decreases as additional delay increases. As shown in Figs. 5 and 6, the non-volatility emulation board can be utilized to quantitatively evaluate the impact of long latency in new memories.

IV. Real PRAM-based Prototyping System

We built another evaluation board to evaluate a real PRAM with Xilinx Zynq XC7Z030, which is almost the same FPGA as XC7Z020 except one more PL bank. Our PRAM-based prototyping board has three different memory subsystems; PS-DRAM, PL-PRAM and PL-DRAM. In this board, PS-DRAM is a four-chip DIMM of 1GB, PL-PRAM a four-chip So-DIMM of 512MB and PL-DRAM a single chip of 256MB. PS-DRAM is the same as the one in the previous non-volatility emulation board. PL-side memories (PRAM DIMM and a DRAM chip) are connected to DRAM and PRAM controllers, and accessed via AXI bus by PS core.

A. PRAM Characteristics

The PRAM chip has LPDDR2-N interface protocol with 16b data [5]. Although it has LPDDR compatible pins, the operation commands are different from those in DRAM. PRAM read is performed with three operations, preactive (PREACT), activate (ACT), and read (RD). Furthermore, PRAM write operation adopts a flash-like two-stage scheme. Thus, the first step writes data to volatile program buffer (WR). The second step moves the data from program buffer to non-volatile PRAM cells (PR). A PRAM chip has an internal 512B program buffer to boost up write operation using embedded write driver. We call PRAM with the two-staged write scheme *buffered PRAM*.

TABLE II. Command Latency to PL-side Memories

Command	Destination	Size [bytes]	Latency [ns]
Read (RD)	PL-DRAM	4	288
Write (WR)	PL-DRAM	4	202
RD	PL-PRAM	4	408
WR	PL-PRAM	4	469
Program (PR)	PL-PRAM	4	7,531
PR	PL-PRAM	2048	79,069

Table II shows the latency of single command in PL-DRAM and PL-PRAM. We use the standalone OS provided by Xilinx to measure the PL-side latency and bandwidth. PRAM has relatively comparable RD latency (408ns) to DRAM (288ns). However, PRAM spends a long latency, ~7.5μs to write 4B data while DRAM takes only 202ns. More data are sequentially gathered, higher write bandwidth is obtained due to optimization of embedded write driver. Program (PR) command becomes more effective as the write data size increases. Thus, 2KB program takes 79μs.

B. PRAM Controller

Un-buffered single chip PRAM (4B data write) scores only 0.5MB/s write bandwidth, on the other hand, buffered single chip PRAM (2KB data write) scores x8 times higher bandwidth, 4MB/s. Note that the write bandwidth of 4MB/s is similar to that of class-4 SD card. In order to utilize the performance benefit of buffered write, we equipped the PRAM controller with an internal program buffer gathers write data belonging to the same unit of program. On receiving write

requests, as far as they go to the same unit of program, the PRAM controller accepts and writes write data to the internal program buffer. When the internal program buffer becomes full or a write request is not associated with the program unit, the memory controller initiates a PR command and writes the contents of internal buffer to the program buffer of PRAM. Note that the larger internal program buffer can give the better performance due to better effects of write merging, which is left for our future work.

The internal program buffer gives another benefit in read operation. Due to the separate read/write paths in the PRAM, when there is a read request to the contents in the program buffer of PRAM chip, without the internal program buffer, the PRAM controller first needs to program the contents of program buffer. Then, the required data can be read from the PRAM chip, which incurs prohibitively long read latency. On the contrary, the internal program buffer can provide the required data thereby avoiding the intermediate long program operation.

C. Cache and PRAM Main Memory

The cache can improve overall performance in the DRAM-based system. However, our PRAM-based main memory has a different tendency. Due to the long write latency, PRAM-based main memory gives a high penalty of dirty cache block eviction. An eviction of one 64B cache block takes 14μs.

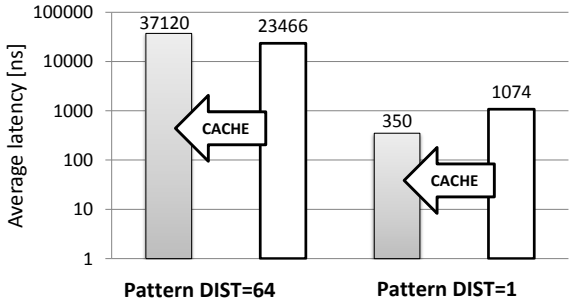


Fig. 7. Latency of write traffics to PRAM-based main memory

Fig. 7 shows the impact of cache eviction on PRAM-based main memory. For each of two traffic patterns (DIST=64 and 1), we compare the average latency of two cases (with and without cache usage). We generate write traffics (requests of 1B write) with the footprint of 64MB. Pattern DIST=64 incurs a lot of cache conflicts each of which takes a long eviction latency, 14μs. Dirty block evictions from L2 cache degrade average access latency by 58% (23466 -> 37120ns). On the contrary, when pattern DIST=1, i.e., sequential access pattern is applied, by utilizing the cache, average latency is improved by 67% since the cache merges write data and reduces PRAM write traffics.

In the hybrid DRAM/PRAM main memory [11][12], PRAM can be utilized in two ways, large background main memory and a partition. When PRAM is utilized as a large background memory, long write latency can be mitigated by DRAM cache. However, when PRAM is used a memory partition, i.e., PRAM is assigned a region of physical memory address, dirty cache evictions to PRAM will incur long write latency as shown in Fig. 7. Thus, software and hardware designs for new memory like PRAM need to take into account long write latency of new memory. Especially when the new memory is utilized to cover a memory partition, i.e., software and hardware designs need to minimize dirty eviction from the on-chip cache to the new memory, e.g., by reducing random writes.

D. Experiments

Table III shows the system configuration of our PRAM-based board. In order to prove the utility of this board, we use a set of Android program execution traces. We developed a trace extraction and replay tool. In order to obtain traces, first, we inserted trace generation functions to SQLite library. While a real user runs an Android application which uses SQLite for storage, the trace generation functions called by SQLite library functions generate traces in two files, i.e., command and data trace files. Then, on our PRAM-based board, we run our trace replay program to access the PRAM subsystem. We used various Android applications; acore, media, browser, facebook, gmail, youtube, kakao talk and twitter. Each application execution takes few seconds to tens of seconds.

TABLE III. SYSTEM PARAMETERS

Component	Parameters
Core	ARM Cortex-A9 dual core, 667MHz
Cache	L1 I/D = 32KB/32KB, L2 = 512KB
PS-DRAM	256MBx4, DDR3-1066, 8bx4, 7-7-7
PL-DRAM	256MBx1, DDR3-800, 8b, 6-6-6
PL-PRAM	128MBx4, LPDDR2-150, 16bx4, 2kB buffered WR

We experiment total 12 traces (Fig. 8) on three different memory subsystems: DRAM, Naïve Hybrid and Persistent. ‘DRAM’ indicates the case of in-memory DB without journaling which is supported by the original SQLite. In this case, both main memory and storage are allocated in DRAM. Thus, SQLite commands are executed fast. However, it is not persistent. System ‘Naïve Hybrid’ combines 256MB PRAM and 512MB DRAM on the contiguous physical address region. PRAM occupies one third of main memory and serves some of memory traffics. In ‘Persistent’, PRAM works as fast non-volatile non-cacheable memory.

A simple integration of PRAM as a part of main memory can significantly degrade overall performance. In our observation, ‘Naïve Hybrid’ has average 4.3x longer total execution time than ‘DRAM’. However, in the case of ‘Persistent’, user can determine data location, i.e., which data to store in fast volatile DRAM or slow persistent PRAM. In our example, we store only DB data (journal and DB file) on PRAM while preventing frequently modified data, i.e. heap, stack and page table from being stored in PRAM. Thus, ‘Persistent’ reduces performance degradation down to 1.8x on average. We expect new memory-aware software/hardware design optimizations can give further reduction in performance degradation.

With application youtube, PRAM reveals severe (10x) performance degradation. Twitter gives relatively small (2x~3x) performance degradation due to its low command intensity, since twitter processes only 62 SQL commands per second. On the other hand, browser-2 has 5.7x performance

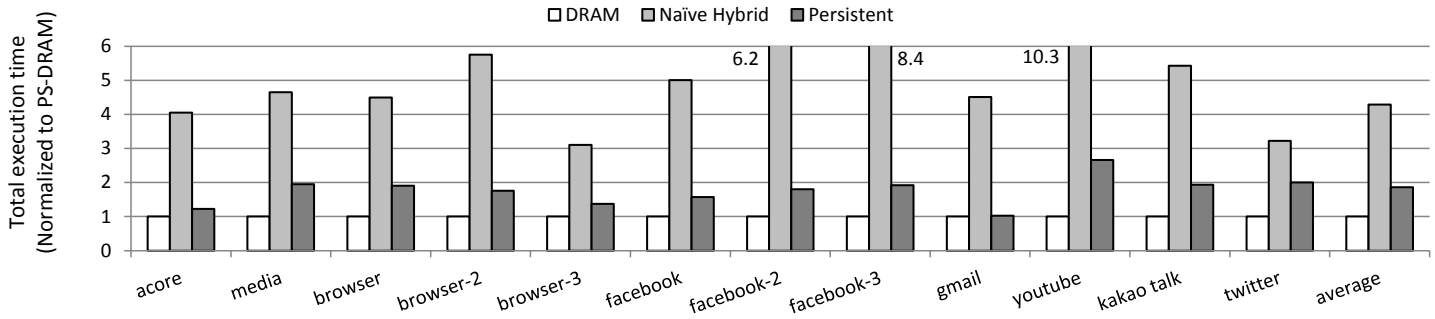


Fig. 8. Execution time of Linux applications

degradation processing 4k commands per second. Application gmail has only 2% performance overhead with 'Persistent'. It is because gmail is read-intensive (most commands are SELECT), and hardly performs DB updates, i.e. INSERT. However, 'Naïve Hybrid' still suffers from 4.5x performance degradation with gmail.

V. Conclusion

We developed two FPGA-based prototyping boards for emerging memory technologies. One, called non-volatility emulation board, has the functionality of non-volatile DRAM utilizing separate power switch to keep DRAM data during system shutdown and is equipped with parameterizable read/write latency to evaluate the impact of lower performance of new memories than DRAM. The other, called real PRAM-based prototyping board, has LPDDR2-N based PRAM and in-house PRAM controller with buffered write capability. We evaluated the performance of primitive functions to understand the performance impact of different subsystems (PS and PL) and cacheable conditions. In addition, we ran real user traces obtained from SQLite-based Android applications and diagnosed the impact of new memory based main memory on software performance.

Acknowledgment

This work was supported by the IT R&D program MKE/KEIT (No.10041608, Embedded System Software for New Memory-based Smart Devices) and Samsung Electronics.

References

[1] International Technology Roadmap for Semiconductors (ITRS), available at www.itrs.net

[2] B. C. Lee, et al., "Phase-Change Technology and the Future of Main Memory," Proc. MICRO, 2010..

[3] Kultursay, E., et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," Proc. ISPASS, 2013.

[4] Tanakamaru, S., et al., "NAND Flash Memory/ReRAM Hybrid Unified Solid-State-Storage Architecture," IEEE Trans. Circuits and Systems I: Regular Papers, vol. 61, issue 4, April 2014.

[5] H. Chung, et al. "A 58nm 1.8V 1Gb PRAM with 6.4MB/s Program BW," Proc. ISSCC, 2011.

[6] D. Kim, et al., "Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU," Proc. DAC, 2012.

[7] P. Zhou, et al., "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," Proc. ISCA, 2009.

[8] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A Hybrid PRAM and DRAM Main Memory System," Proc. DAC, 2009.

[9] Qureshi, M. K., et al., "Enhancing lifetime and security of PCM-based Main Memory with Start-Gap Wear Leveling," Proc. MICRO, 2009.

[10] J. Coburn, et al., "NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories," Proc. ASPLOS, 2011.

[11] M. K. Qureshi, V. Srinivasan, J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," Proc. ISCA, 2009.

[12] H. Park, S. Yoo, S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," Proc. DAC, 2011.

[13] Y. Jin, et al., "Area, Power, and Latency Considerations of STT-MRAM to Substitute for Main Memory," Proc. ISCA, 2014.

[14] E. Lee, et al., "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory," Proc. FAST, 2013.

[15] A. Akel et al. "Onyx: A Prototype Phase Change Memory Storage Array," Proc. HotStorage, 2011.

[16] A. M. Caulfield, et al., " Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories," Proc. MICRO, 2010.

[17] B. C. Lee, et al. "Architecting phase change memory as a scalable dram alternative," Proc. ISCA, 2009.