

# A Case for Small Row Buffers in Non-Volatile Main Memories

Justin Meza\* Jing Li† Onur Mutlu\*

\*Carnegie Mellon University †IBM T.J. Watson Research Center  
 {meza,onur}@cmu.edu jli@us.ibm.com

**Abstract**—DRAM-based main memories have read operations that destroy the read data, and as a result, must buffer large amounts of data on each array access to keep chip costs low. Unfortunately, system-level trends such as increased memory contention in multi-core architectures and data mapping schemes that improve memory parallelism lead to only a small amount of the buffered data to be accessed. This makes buffering large amounts of data on every memory array access energy-inefficient; yet organizing DRAM chips to buffer small amounts of data is costly, as others have shown [11].

Emerging non-volatile memories (NVMs) such as PCM, STT-RAM, and RRAM, however, do not have destructive read operations, opening up opportunities for employing small row buffers without incurring additional area penalty and/or design complexity. In this work, we discuss and evaluate architectural changes to enable small row buffers at a low cost in NVMs. We find that on a multi-core system, reducing the row buffer size can greatly reduce main memory dynamic energy compared to a DRAM baseline with large row sizes, without greatly affecting endurance, and for some NVM technologies, leads to improved performance.

## I. INTRODUCTION

Modern main memory is composed of dynamic random-access memory (DRAM). A DRAM cell stores data as charge on a capacitor. Over time, this charge leaks, causing the stored data to be lost. To prevent this, data stored in DRAM must be periodically read out and rewritten, a process called *refreshing*. In addition, reading data stored in a DRAM cell destroys its state, requiring data to be later restored, leading to increased cell access time and energy. For this reason, DRAM devices require *buffering* data which are read. To keep costs low, the buffering circuitry in DRAM devices is amortized among large *rows* of cells, in peripheral storage called the *row buffer*, at least one per bank [2]. Refreshing data and buffering large amounts of data wastes energy in DRAM devices, causing main memory power to constitute a significant fraction of the total system power.

Data fetched into the row buffer, however, can be accessed at much lower latencies and less energy than accessing the DRAM memory array. Therefore, large row buffer sizes can improve performance and efficiency if many accesses can be served in the same row. Unfortunately, there are several reasons why such *row buffer locality* can be low in systems: (1) some applications inherently do not have significant locality within rows (e.g., random access applications), (2) as more cores are placed on chip, applications running on those cores interfere with each other in the row buffers, leading to reduced locality, especially if the memory scheduling policy is unaware of applications' interference in the row buffers [7], as also observed by others [10, 11], and (3) interleaving techniques that improve parallelism in the memory system (e.g., cache block interleaving) tend to reduce row buffer locality because they stripe consecutive cache blocks across different banks. As a result, the performance benefit of large row buffers may decrease in multi-core systems.

New non-volatile memory (NVM) technologies, such as phase-change memory (PCM), spin-transfer torque RAM (STT-RAM), and resistive RAM (RRAM), on the other hand, provide non-destructive reads and do not require refreshing and restoring their data after sensing. This is because NVMs do not store their data as charge, and thus their data persists after being read. This not only eliminates the refresh problem of DRAM devices but also opens up opportunities for employing smaller row buffers in NVMs without incurring additional area penalty and/or design complexity.

## II. MOTIVATION

Emerging NVM technologies have several promising attributes compared to existing memory technologies such as SRAM (used in

on-chip caches), DRAM, and Flash. For example, NVMs provide cost advantages compared to SRAM and DRAM, and latency advantages compared to Flash. Importantly, these NVMs feature *non-destructive read operations*, which DRAM does not have (i.e., data sensing does not destroy the contents of cells).

Typical DRAM chip micro-architectures (JEDEC-standard DDR-type SDRAM) are divided into banks that consist of rows (wordlines) and columns (bitlines). Due to physical pin limitations, all the information required to service a memory request must be supplied over multiple commands. The *Row Address Strobe (RAS)* command sends the row and bank address to select one of the banks and a row within that bank. Then, an entire row (usually 1 to 2KB per chip) is read out into latch-based sense amplifiers which comprise the row buffer [2]. The *Column Address Strobe (CAS)* command then selects a subset (i.e., column) of data from the row buffer (8B in a DDR3  $\times 8$  device [6]). Thus, a DRAM access first fetches many kilobytes of data into the row buffer (RAS) and, in the worst case, uses only a tiny portion of it (CAS). If multiple columns of the row buffer are needed, multiple consecutive CAS commands can be issued, which amortizes the cost of fetching the large row into the row buffer.

To illustrate how much buffered data is actually used in real applications, Figure 1a shows average row buffer locality (row hit rate) when employing various row buffer sizes on several system configurations using the FR-FCFS scheduling policy [8].<sup>1</sup> In particular, we show 1- and 8-core systems employing two different schemes for mapping data in main memory: (1) *row interleaving*, which places consecutive memory addresses in the same row, and (2) *block interleaving*, which stripes data in consecutive memory addresses (usually cache blocks) across different banks. Row interleaving helps exploit row buffer locality by enabling data with spatial locality to reside in the same row buffer, while block interleaving aims to improve memory parallelism by enabling concurrent access of memory channels/banks for consecutive memory addresses.

Comparing the 1- and 8-core row-interleaved data, we see that while row interleaving does enable more row buffer locality, its benefits diminish as memory system contention increases with more cores: row buffer hit rate is less than 50% for row interleaving even with large, 1KB rows. Block interleaving reduces row buffer locality over row interleaving, to less than 10% in the 8-core case. While it is clear that row locality is lower on multi-core systems, what is less obvious is how row buffer size affects system-level tradeoffs, such as energy-efficiency, performance, and durability, in NVM main memories. This work evaluates these tradeoffs.

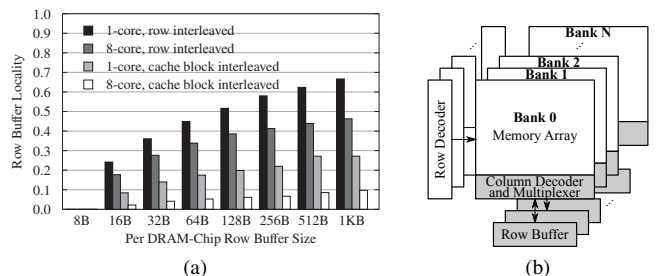


Fig. 1: Row size affects row locality (a); our NVM architecture (b).

<sup>1</sup>Application-aware memory request scheduling policies (e.g., [1, 3, 7]) provide better performance, but they can reduce row buffer locality.

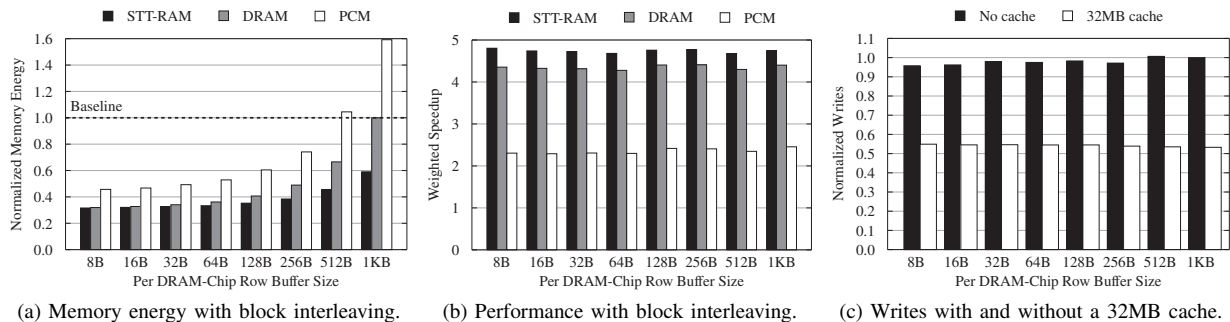


Fig. 2: Multi-core results for energy (normalized to DRAM with 1KB rows), performance, and number of writes (normalized to 1KB rows).

### III. A SMALL ROW BUFFER NVM ARCHITECTURE

Figure 1(b) shows the organization of our NVM architecture. Compared to a traditional DRAM organization, the physical placement of the row buffer and the column multiplexer (part of the I/O gating circuitry in DRAM designs) are swapped in the data path (shown in gray). This rearrangement makes better use of resources by sharing a smaller number of *sense amplifiers* (the devices which store bits in the row buffer) among multiple bitlines. Note that this is not possible in DRAM (without reducing the row size) because a sense amplifier for each bit in the row is required in DRAM to restore the charge of the cell after it is read. Unlike DRAM, however, our organization requires decoding both the row address *and* the column address during a *RAS* command, so that only a subset of the row containing the bits of interest will be selected, sensed, and stored in the row buffer. During a *CAS* command, the data bits from the row buffer corresponding to the desired column are further selected by the I/O gating circuitry and sent to a prefetch buffer.<sup>2</sup>

While related prior work [4] employed multiple, narrow rows in a PCM main memory for reducing array reads and writes, it focused on (1) a traditional DRAM data path design, (2) an iso-area reorganization, requiring more area overhead than our technique which employs smaller row buffers, and (3) assumed a standard DRAM protocol for device access.

### IV. RESULTS

We developed a cycle-accurate DDR3 memory simulator which we use as part of an in-house x86 multi-core simulator, whose front-end is based on Pin. We modify our memory simulator timings according to those in Table I for PCM and STT-RAM. We show results for an 8-core system with different memories and row buffer sizes, though reducing row buffer size in DRAM incurs significant area overhead and chip cost, as discussed in [10, 11], which we do not evaluate. We evaluate 31 multiprogrammed workloads composed of SPEC, TPC, and STREAM benchmarks. We will focus on a DRAM chip micro-architecture with 1KB row buffers and block interleaving as our baseline (our findings are similar for row interleaving [5]).

Technology	Energy (Read/Write)	Latency (Read/Write)
PCM	2×/100×	5×/10×
STT-RAM	0.5×/1×	1×/1×

TABLE I: NVM array parameters, relative to DRAM.

**Energy (Figure 2a):** In all cases, reducing the row buffer size can significantly reduce memory energy consumption, though there are diminishing marginal returns. The diminishing marginal returns are because, as the row buffer size decreases, memory energy becomes dominated by the energy required to transfer data between the row buffer and I/O pads during read and write operations.

A modest row buffer size of 64B per chip leads to 47%/67% less main memory energy for PCM/STT-RAM, compared to an all-DRAM main memory with large rows (1KB per chip). Note that this reduction is achieved despite worse underlying technology parameters

<sup>2</sup>For more details, please refer to our accompanying tech report [5].

than DRAM (cf. Table I) because the energy saved by reducing the row buffer size more than makes up for the higher average memory array access energy. Hence, an NVM main memory with smaller row buffers can significantly reduce memory energy consumption compared to a DRAM baseline with large row buffers.

**Performance (Figure 2b):** We evaluate the performance of our system using the weighted speedup metric [9] (higher is better). For a given memory technology, reducing the row buffer size does not greatly affect system performance due to the already low row buffer locality present on our multi-core system (cf. Figure 1a). Interestingly, with similar technology-dependent timing parameters as DRAM, an STT-RAM main memory can achieve better performance because our new organization enables a more efficient access protocol (detailed in [5]) which eliminates the precharge delay incurred on row buffer misses, and relaxes the  $t_{RRD}$  and  $t_{FAW}$  timing parameters to enable more banks to be accessed simultaneously.

**Durability (Figure 2c):** NVM cells have a limited lifetime in terms of the number of times they can be written to before their ability to store data fails. We examine the effects of different row buffer sizes on device durability with and without a small 32MB e-DRAM cache to a PCM main memory. We find that with or without a cache, decreasing the row buffer size has only a small effect on the number of NVM writes performed due to the low row buffer locality present in the system. In contrast, the addition of a reasonably-sized e-DRAM cache has a large impact on the reduction of writes, decreasing the number of writes by 39% to 47% across the various row buffer sizes.

### V. CONCLUSIONS

We showed that on a multi-core system, reducing the row buffer size can greatly reduce main memory dynamic energy compared to a DRAM baseline with large rows, without greatly affecting performance and durability. Our future work includes exploring architectural techniques which effectively leverage small row buffer sizes for improved performance and energy-efficiency.

### REFERENCES

- [1] Y. Kim et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. HPCA '10.
- [2] Y. Kim et al. A case for exploiting subarray-level parallelism (SALP) in DRAM. ISCA '12.
- [3] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. MICRO '10.
- [4] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. ISCA '09.
- [5] J. Meza, J. Li, and O. Mutlu. A case for small row buffers in non-volatile main memories. <http://safari.ece.cmu.edu/tr-tr-2012-002.pdf>.
- [6] Micron. 1Gb: ×4, ×8, ×16 DDR3 SDRAM data sheet. <http://download.micron.com/pdf/datasheets/dram/ddr/1GbDDR4x8x16.pdf>.
- [7] S. P. Muralidhara et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning. MICRO '11.
- [8] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. ISCA '00.
- [9] A. Snaveley et al. Symbiotic jobscheduling for a simultaneous multi-threading processor. ASPLOS '10.
- [10] K. Sudan, N. Chatterjee, D. Nellans, et al. Micro-pages: increasing dram efficiency with locality-aware data placement. ASPLOS '10.
- [11] A. N. Udipi, N. Muralimanohar, N. Chatterjee, et al. Rethinking DRAM design and organization for energy-constrained multi-cores. ISCA '10.