Theme Article

CONCEPT: A Column-Oriented Memory Controller for Efficient Memory and PIM Operations in RRAM

Nishil Talati
University of Michigan, Ann Arbor
and Technion—Israel Institute of Technology

Heonjae Ha
Stanford University

Ben Perach
Technion—Israel Institute of Technology

Ronny Ronen
Technion—Israel Institute of Technology

Shahar Kvatinsky
Technion—Israel Institute of Technology

***Abstract*—While DRAM cannot easily scale below a 20-nm technology node, RRAM suffers far less from scalability issues. Moreover, RRAM's resistivity enables its use for processing-in-memory (PIM), potentially alleviating the von Neumann bottleneck. Unfortunately, because of technological idiosyncrasies, existing DRAM-centric memory controllers cannot exploit the full potential of resistive RAM (RRAM). In this paper, we present the design of a memory controller called CONCEPT. The controller is optimized to exploit unique properties of RRAM to enhance its performance and energy efficiency as well as exploiting RRAM's PIM capability. We show that with CONCEPT, RRAM can achieve DRAM-like performance and energy efficiency on SPEC CPU 2006 benchmarks. Furthermore, using RRAM PIM capabilities, we show a 5× performance gain on a data-intensive in-memory database workload compared to a state-of-the-art CPU-memory computing model.**

Digital Object Identifier 10.1109/MM.2018.2890033
Date of publication 3 January 2019; date of current version 21 February 2019.

The process technology scaling of DRAM has so far facilitated low cost-per-bit by enabling reduction in cell size. However, further scaling has proven to be costly due to physical factors, such as the difficulty of fabricating capacitors

0272-1732 © 2019 IEEE

with high aspect ratio. Furthermore, because commodity DRAM cannot process data, it needs to be transferred to the CPU for processing via a bandwidth-limited bus. This data transfer severely limits the performance and the energy efficiency of computers,[1] a limitation known as *the von Neumann bottleneck*.

The memory controller is a key component in optimizing the performance and energy efficiency of modern computing systems. Its purpose is to deliver performance and power optimizations by intelligently scheduling memory requests and accessing data given the limitations of a memory technology. DRAM has been the technological choice for building main memory systems for several decades mainly because of its low access latency and low cost. Therefore, existing memory controllers target DRAM, and their design constraints are focused around that specific technology.

To evaluate our proposed system, we develop a comprehensive device-to-architecture simulation framework. Using this simulation framework, we compare performance and energy efficiency of our memory system against RRAM accessed using DDR4 on SPEC CPU2006 workloads.

In this paper, we propose to employ resistive RAM (RRAM) to tackle both the DRAM scaling problem and the von Neumann bottleneck. While prior art employs a traditional DRAM memory controller to leverage unique properties of RRAM,[7] we propose a more fundamental and novel solution to the problem: redesigning the memory controller to optimize the data access pattern as well as to perform processing-in-memory (PIM) using RRAM.

We make several key observations about the technological idiosyncrasies that make DRAM and RRAM incompatible. Motivated by these observations, we propose a unique column-oriented access protocol for RRAM, which we call R-DDR, i.e., an RRAM-access protocol similar to DDRx. This protocol is tailor made around the design constraints of RRAM to best optimize its performance and energy efficiency. Optimizations proposed in the past[7] are orthogonal to our approach, which can be augmented on top of our approach, which may lead to further benefits in performance and energy efficiency of RRAM. In addition to supporting efficient memory operations, R-DDR can also support PIM operations using a technique called Memristor-Aided loGIC (MAGIC).[2] MAGIC-based PIM allows reducing data transfer to the CPU and leveraging the ample parallelism of the memory crossbar array[3] to realize efficient single instruction multiple data (SIMD) operations. We extend the instruction set architecture (ISA) and hardware of the memory controller to support two additional instructions for PIM—two-input MAGIC NOR and MAGIC NOT. Note that our solution is motivation from how DDRx is highly optimized for a technology (i.e., DRAM); however, we believe that in the future with a heterogeneous memory system, a programmable controller might be useful to support various protocols optimized for different memory technologies.

To evaluate our proposed system, we develop a comprehensive device-to-architecture simulation framework. Using this simulation framework, we compare performance and energy efficiency of our memory system against RRAM accessed using DDR4 on SPEC CPU2006 workloads. The results show that our approach improves performance by 35% and reduces energy by 7%. These improvements are the result of carefully designing the memory controller around the constraints of RRAM to optimize its memory access pattern. A comparison of our approach with other PIM approaches shows that our system outperforms PCM-based Pinatubo[5] by 5× and achieves similar performance to DRAM-based Ambit.[4]

BACKROUND AND MOTIVATION

DRAM Scaling Problem

DRAM uses a capacitor to store charge that is accessed by an access transistor. This capacitive element is generally fabricated in a vertically cylindrical structure to save chip area. To maintain the retention time requirement of the DRAM device, the capacitance needs to be constant at approximately 25 fF. DRAM scaling has so far been facilitated by scaling a linear dimension by 0.71× from generation to generation, and hence, the surface area of the capacitor scales by 0.5×. To compensate for the reduction in

Published by the IEEE Computer Society

surface area, the height of the capacitor needs to be scaled by 2× to keep the capacitance value the same. The aspect ratio of the capacitor, which is the ratio of the height and its base diameter, increases exponentially with technology scaling, and goes above 100× at the sub-20-nm technology node; hence, DRAM companies use costly sophisticated tricks to get around this issue. These narrow cylindrical cells are inherently unstable and mechanically difficult to fabricate. On the other hand, if the capacitor height is not scaled to compensate for the increase in area, the capacitor can store comparatively less charge. As a result, DRAM refresh rate must be increased, reducing the performance and increasing the energy consumption of the memory system. Therefore, the scaling of DRAM is a major challenge below the 20-nm technology node.

Von Neumann Bottleneck

Contemporary computing systems are based on the von Neumann architecture, or an improvised version of it, where data are processed within the CPU and stored in memory (DRAM being main memory). Since traditional DRAM cells are charge based and they cannot process data using the same cells that are used for storing data, data must be transferred outside the memory bank or chip for processing. The resulting data movement between the memory and the CPU causes major bottlenecks in terms of performance and energy efficiency.[1] A few contemporary approaches propose to use DRAM sense amplifiers for computation. However, doing so would still require data movement from the memory cells to the sense amplifiers and back; hence, they are limited by the bandwidth available for data transfer between cell array and peripheral circuits. One example of such an approach is Ambit.[4]

RRAM and Data Access in RRAM

RRAM is a nonvolatile memory technology that can store a logical value in terms of its resistance. Typically, an RRAM device is fabricated using a metal–insulator–metal (MIM) structure, where the insulator is rich in oxygen ions. The voltage across the RRAM device controls the drift of the oxygen ions, allowing the formation/rupture of conducting filaments, and thereby forming distinct resistance states. The small diameter of the conducting filaments makes RRAM technology *extremely scalable*, and it is thus an attractive candidate for a future low-cost memory technology. Conventionally, RRAM has two resistance states: the low-resistance state, which represents logical 1, and the high-resistance state, which represents logical 0.

The write operation in RRAM is performed in two phases by applying write voltages via wordlines (WLs) and bitlines (BLs)—SET (to write 1s) and RESET (to write 0s). Read operations in RRAM are performed by applying a voltage pulse of typically lower magnitude than the write voltage and measuring the current flowing out of the cell using current-based sense amplifiers. In addition to performing memory operations, RRAM can also execute logical functions by realizing unique circuit connections and applying voltages. In this paper, we focus on MAGIC[2] to perform computation using RRAM cells. Advantages of MAGIC include crossbar compatibility, statefulness (which enables true PIM),[6,11] functional completeness, and parallel execution.[6] Bitwise SIMD MAGIC NOR and NOT operations can be performed inside RRAM to realize PIM merely by applying a voltage pattern, and without any modification of the conventional memory crossbar structure.

Why a New Memory Controller for RRAM?

RRAM and DRAM differ in various ways that affect the memory controller.

- *Row-oriented versus column-oriented access mechanism*: The DRAM access mechanism is row oriented, where an access starts with opening a memory row and reading its data to the row buffer. Note that this operation is performed irrespective of whether the request type is read or write. In contrast, RRAM uses different system resources for different operations (i.e., voltage drivers during write versus voltage drivers + sense amplifiers during read) as well as unique biasing schemes (i.e., half-select during write versus floating during read). This means that *the type of access* must be known in advance (which is sent using the column command). Hence, the RRAM access protocol needs to be column oriented

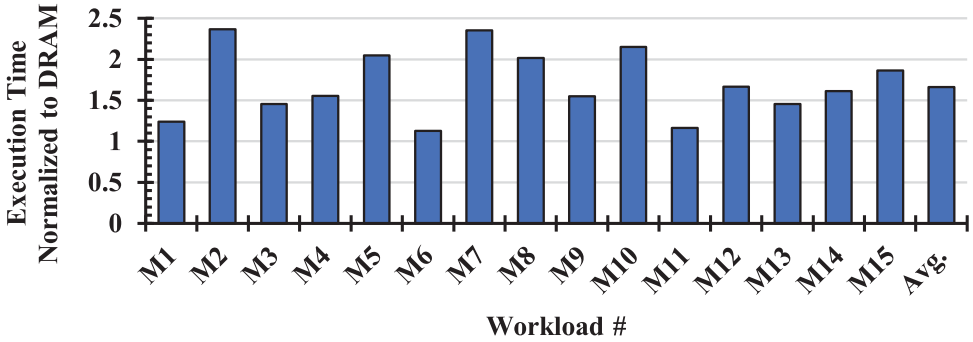

Figure 1. Execution time of RRAM accessed using the DDR4-2400-based memory controller normalized to DRAM accessed using a similar memory controller.

(meaning centered around read/write (column) command rather than a row command) as opposed to row oriented in DRAM. Note that while we propose to change what RRAM protocol should be centered around, each row access (i.e., opening a memory row and reading data to the row buffer) and column access (i.e., reading/writing to few columns) have the same meaning as DDRx.

- *Open-page versus closed page policies*: RRAM multiplexes multiple BLs to a single sense amplifier to reduce the cost of the chip.[5] As a result, unlike DRAM, data from a full row are not available at the row buffer during a read operation. Consequently, the RRAM memory controller cannot employ an open-page policy, and generally, a restricted closed-page policy is used (i.e., closing the row immediately after an access is serviced).
- *Refresh and restore*: A few DRAM operations, namely restore and refresh, are not required in RRAM because of nondestructive reads and high retention times of RRAM.
- *PIM support*: Since DRAM cells cannot perform PIM, the traditional memory controller does not support PIM instructions. Thus, to exploit PIM capabilities of RRAM, we need to support PIM instructions for the RRAM memory controller.

To demonstrate the disadvantages of accessing RRAM using the DRAM protocol, we model an RRAM-based main memory as described in the methodology section, and access it using DDR4-2400. Figure 1 shows the execution time for different workloads listed in Table 1. For these experiments, we use a restricted closed-page policy and the same auxiliary timing constraints as DDR4 (i.e., tCCD, tRRD, tFAW, and tWRT).[7] We model RRAM read and write latencies by tCAS and tWR, respectively. RRAM accessed using a DDRx-based memory controller is 66.3% slower than DRAM. Additionally, we will show in the next sections that the access protocol also affects these values. Finally, we will show how customizing the memory controller for RRAM can unleash its full potential.

Table 1. SPEC CPU 2006 workload setup.

#	MPKI	Benchmark Mixture
M1	Low-Low-Mid-High	povray-gromacs-soplex-cactusADM
M2		GemsFDTD-gamess-wrf-milc
M3		hmmer-h264ref-perlbench-mcf
M4		sphinx3-namd-gcc-lbm
M5		tonto-dealII-bwaves-leslie3d
M6	Low-Mid-Mid-High	calculix-libquantun-zeusmp-cactusADM
M7		xalancbmk-soplex-wrf-milc
M8		gobmk-wrf-perlbench-mcf
M9		sjeng-gcc-bwaves-lbm
M10		omnetpp-libquantum-zeusmp-leslie3d
M11	Low-Mid-High-High	bzip2-soplex-cactusADM-milc
M12		astar-wrf-mcf-lbm
M13		povray-perlbench-leslie3d-cactusADM
M14		gromacs-gcc-milc-mcf
M15		GemsFDTD-bwaves-lbm-leslie3d

CONCEPT DESIGN

In this section, we propose a memory controller

Published by the IEEE Computer Society

to perform efficient memory as well as PIM operations in RRAM. The controller is divided into the front-end (responsible for receiving and storing the memory requests and PIM commands such as MAGIC NOR/NOT) and the back-end (responsible for converting requests into low-level technology-specific commands and sending these commands to access main memory data while maintaining timing constraints). We present microarchitectural modifications of transaction queues to support new PIM instructions. Then, we discuss our proposed RRAM access protocol and the scheduling algorithm.

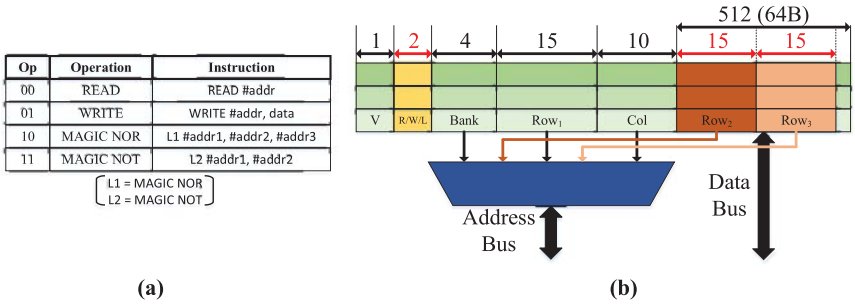

Figure 2. (a) ISA extension of the memory controller to support PIM instructions using MAGIC. (b) Modified design of the transaction queue that can also support PIM instructions. The different field lengths correspond to 4 Gb x8 DDR4 DRAM. Legends: V—valid; Op—opcode. Bit widths in red font represent the proposed extension from a traditional transaction queue design.

CONCEPT Front-End Design

On a last-level cache miss, a request is transferred to the memory controller to perform a memory transaction, and the front-end of the memory controller is responsible for receiving this request. First, the physical address is decoded to a memory-specific coordinate in terms of channel, rank, bank, row, and column. In this paper, we assume a traditional scheme of address decoding where upper bits of the physical address are mapped to rows to exploit row-buffer locality, lower bits are mapped to columns, and middle bits are mapped to bank addresses. We then propose changes to modify the transaction queue structure.

Figure 2(a) shows the extended ISA of our memory controller. The first two entries in the table are conventional read and write instructions—a read instruction with an address field and a write instruction with an address as well as a data field. The last two entries are the MAGIC instructions to perform SIMD NOR and NOT operations. These instructions are unique in that they use a higher number of addresses than read/write without data. MAGIC NOR instruction uses three addresses—two for inputs and one for output. MAGIC NOT instructions use two addresses only—one for inputs and one for output. To incorporate PIM instructions in transaction queues, the fields of the transaction queue can be naively extended to support three addresses independently. However, this solution would incur significant hardware overhead, and thus is impractical. We propose to incorporate the PIM instruction format in transaction queue structure with minimum area overhead. Our proposal is based on four key observations.

- To reduce the pin count (and hence, the packaging cost), the memory controller sends row and column addresses in a time-multiplexed fashion.
- The data field is much wider (i.e., 512 bits) than the address field (i.e., 15 bits for a row address).
- MAGIC operations can be performed only on vectors present in the same RRAM bank (at row granularity), so only one bank address is required per instruction.
- PIM instructions consist only of addresses and not data since they are performed on operands already situated inside the memory.

We, thus, propose to reuse a few of the data-field bits to create a complete MAGIC instruction within the queue without significantly increasing storage space. Figure 2(b) shows the design of a modified transaction queue where two row addresses are replicated within the unused data field and a path to the multiplexer is created for time-multiplexing them. Here, we assume that the PIM instructions are executed row wise; hence, three (two) row addresses and one bank are required to execute a MAGIC NOR (NOT) instruction on a bit-vector present across a whole row. Although PIM operations can be

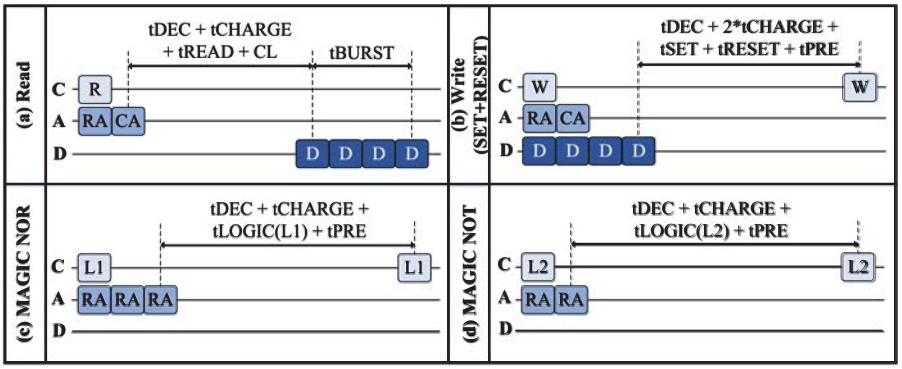

Figure 3. Timings and bus activities of different operations in R-DDR. (a) Read operation. (b) Write operation (composed of SET and RESET phases). (c) MAGIC NOR (L1) operation. (d) MAGIC NOT (L2) operation.

performed at finer granularity by selecting/deselecting different columns/rows for execution,[3] this approach complicates the control logic and peripheral circuit, and hence, in this paper, we focus on exploiting full parallelism from a row, assuming massive parallelism requirement from the workload. Note that even if PIM accesses are performed using more row address bits, it is always possible to accommodate two row addresses in the wide data field. This will minimize the hardware overhead of incorporating PIM instructions with more than one address. The only insignificant overhead is because of a somewhat complex instruction decoder, an additional bit for two-bit opcode [see Figure 2(b)], and decoding logic for the multiplexer.

CONCEPT Back-End Design

Once the memory requests are latched at the transaction queue, the back-end of the memory controller is responsible for splitting the requests into technology-specific commands and sending these commands at specific time intervals, in accordance with the timing constraints. In this section, we present the R-DDR protocol: an RRAM access protocol similar to DDRx, which can incorporate both memory and PIM instructions.

R-DDR: An RRAM Access Protocol Similar to DDR

Since read, write, and logic operations utilize unique resources on the memory die, we propose a column-oriented protocol R-DDR to access RRAM instead of a row-based protocol. To service a read request from the queue using this protocol, the memory controller sends a read command first rather than sending a generic command like *activate*. This prepares the memory to initiate a memory request since different hardware blocks need to be invoked for different requests. Furthermore, since both row and column addresses are required to be sent, R-DDR sends both the addresses in a time-multiplexed fashion one after the other to avoid the pin-count increase. This is necessary to bias the correct WLs/BLs for write and logic operations, as well as to multiplex the correct BLs to the sense amplifiers. Once the row and column addresses have been sent along with the type of access, the memory starts servicing the access.

When no memory request is being serviced, we say that R-DDR is in the idle state. To eliminate leakage power in the idle state, we choose to bias all the WLs/BLs to a ground voltage so that no current flows through the arrays and data does not get corrupted because of voltage jitters.

Read operations in R-DDR are performed by activating the target WL with a read voltage and multiplexing target BLs to be read to sense amplifiers by means of a column address. Hence, current flows from the driver and RRAM cells to the sense amplifiers, where it is converted to a full logical value. Figure 3(a) shows the timing diagram of a read operation where there are four latencies associated with reading data out from the memory cells and transferring it to the I/O pads once all the addresses are received:

1) address decoding latency (i.e., tDEC);
2) *RC*-delay of wires to bias WL to target voltage (i.e., tCHARGE);
3) sampling and reading current flowing through RRAM via sense amplifiers (i.e., tREAD);
4) the latency to transfer data from the sense amplifiers to the I/O pads (i.e., CL/tCAS, which is like DRAM).

After a time delay equal to the sum of these four latencies, we can expect a data burst to

Published by the IEEE Computer Society

start at the data bus, after which, it takes four clock cycles (i.e., tBURST) for all 64B to get transferred to the CPU.

Write operations in R-DDR are performed by activating target WLs and BLs to write voltages in two steps[7]—SET and RESET. The R-DDR protocol initiates a write request by sending the write command along with row and column addresses and the data that need to be written. The timing diagram of write operations is shown in Figure 3 (b). There are five latencies associated with write operations:

1) address decoding latency (i.e., tDEC);
2) the *RC*-delay to bias selected WLs and BLs for SET process (i.e., tCHARGE);
3) the delay to SET the device itself (i.e., tSET) and charge the array again for the RESET operation (i.e., tCHARGE—required since SET and RESET operations are performed using different voltage levels and polarity and they are applied to different cells);
4) the time to RESET the device (i.e., tRESET);
5) finally, the delay to precharge the array to the idle state to prepare for next access (i.e., tPRE).

Logic operations in R-DDR are performed like write operations. As shown in Figure 3(c) and (d), a MAGIC NOR (NOT) operation is initiated by sending a command and three (two) addresses for inputs and outputs. The delay associated with this operation is comprised of the delay to decode addresses (i.e., tDEC), the *RC*-delay to charge wires to appropriate values (i.e., tCHARGE), time to perform logic itself (i.e., tLOGIC), and the delay to precharge the array to the idle state to prepare for the next operation.

Consecutive requests of the same type to different rows in a bank are handled by first closing and precharging the row of the preceding request to service the next one. A more interesting case occurs when two consecutive requests access the same row. In DRAM, both reads and writes are performed using a row buffer that is generally much wider than a single access. Because of this overprovisioning, once the data are read at the row buffer (which incurs high latency of tRCD), subsequent requests to the same row can be serviced with a much lower latency (i.e., tCCD). Conversely, in RRAM, row buffer is only used in read operations. In case of back-to-back write and logic requests to the same bank, even if they access the same row/column, the array needs to be precharged to service the next request. Moreover, although row buffers are used in reads, not a full row is read out because of bulky sense amplifiers, and hence, most RRAM designs immediately precharge a row to service next read. However, there is still some overprovisioning while reading data out. We propose to exploit this limited locality to service multiple read requests to the same row with lower latency when they hit on the narrow row buffer.

Consecutive requests of different types: When a read request follows a write request, DRAM specifies a timing constraint called tWTR, i.e., a write-to-read bus turn-around delay. Although separate I/O paths exist for read and write requests, tWTR is nonzero because of internal bus and row buffer sharing between read and write requests. This gives rise to a long write-to-read turnaround time (i.e., 7.5 ns), which reduces the DRAM bandwidth. There are two ways to address this problem: by having independent GIO (global I/O) buses for reads and writes that can reduce tWTR, or by buffering write requests to clear the way for reads. The latest DRAM architectures (such as HBM2—high bandwidth memory) use independent I/O lines—RGIO and WGIO—to increase core frequency, and we also expect RRAM vendors to adopt such a design to alleviate bottlenecks due to tWTR. Nevertheless, even if this is not the case, internal bus contention can be avoided by buffering write requests on the memory chip to delay the write request and allow performance-sensitive read requests to get serviced.

To exploit internal parallelism, DRAM can access multiple banks in parallel concurrently. Although such parallelism is desirable, it increases the amount of current required to service many requests, increasing the power consumed in the chip. Furthermore, since the memory market is extremely cost sensitive, DRAM is fabricated using a process with only three metal layers, which can only withstand a certain amount of power. Therefore, in practice, parallelism in DRAM is regulated by two timing parameters—tRRD (i.e., row-to-row delay) and tFAW (i.e., four-bank activation window). To control instantaneous peak power at any time, activate commands to different banks are separated by tRRD. Furthermore, tFAW is defined to specify a rolling time-

Table 2. Parameter values used for simulation.

Parameter	Value	Parameter	Value
V_READ	0.7V	tREAD	11nCK
V_SET	1.4V	tSET	27nCK
V_RESET	1.4V	tRESET	27nCK
V_MAGIC_NOR	1.8V	tMAGIC_NOR	35nCK
V_MAGIC_NOT	2V	tMAGIC_NOT	35nCK
E_READ (array)	0.84pJ/bit	E_READ (periphery)	4.8pJ/bit
E_SET	6.9pJ/bit	E_SET (periphery)	4.8pJ/bit
E_RESET	6.9pJ/bit	E_RESET (periphery)	4.8pJ/bit
Roff	175kOhm	Ron	7kOhm
tDEC	1nCK	tPRE	1nCk
tCHARGE	1nCK	CL	17nCK
tDEC	1nCK	tBURST	4nCK
tWTR	0nCK	tFAW	16nCK
nCK	0.833ns	tRRD	4nCK

frame in which a maximum of four banks can be activated concurrently. This is to control the current drawn from the source in a time window of tFAW, and generally, tFAW>4*tRRD. Values of these parameters change according to the geometry of the DRAM device, which changes the activation current. RRAM can be fabricated in metal layers on a standard logic process and trades off cost with performance by not requiring transistors as access elements. We expect the logic layer in RRAM to support much higher power as compared to a cheap DRAM process, and as a result, the tFAW to be constraint relaxed (i.e., tRRD = 4*tFAW) in RRAM.

Although implementing these optimizations in RRAM incurs an increased cost, this cost is marginal considering the benefits that would be obtained by achieving DRAM-like performance in RRAM. Furthermore, the cost-per-bit for RRAM is lower than for DRAM. While it is also possible to perform power and parallelism-related optimizations to improve its performance in DRAM as mentioned above, we believe there is no justifiable reason for doing so, given the increased cost, and given that DRAM performs well without these optimizations.

EXPERIMENTAL METHODOLOGY

To demonstrate the benefits of our memory design, we compare the performance and energy consumption of our memory system with DDR4-accessed RRAM and DDR4-2400 DRAM. Furthermore, to present the benefit of supporting MAGIC execution from the memory controller, we compare our PIM design with DDR4-2400 based CPU model, and other DDRx-based similar PIM designs, i.e., Pinatubo,[5] and Ambit.[4]

We choose a HfOx-based device[8] for our comparison of RRAM and use VTEAM device model[12] to fit its parameters to the experimental data. Using the voltage levels listed in Table 2, we estimate the array-level latency and energy of read, write, and MAGIC operations using SPICE. Furthermore, we use NVSim[13] to determine the latency and energy of peripheral circuit (which also includes transferring data to and from sense amplifiers and I/O pads).

To evaluate memory operations, we use NVMain.[14] We use SniperSim to gather the main memory traces for the SPEC CPU2006 benchmark suite for 500 M instructions, after 100 M instructions to warm up the cache. The CPU consists of a 4-core, 2.4 GHz Nehalem processor with 2 MB/core, 16-way LRU LLC. DRAM standard is DDR4-2400 17-17-17,[10] with a capacity of 4 Gb with a single channel, a single rank, 16-banks, and 8 KB page size. The workload setup used to evaluate memory system is listed in Table 1. Note that nCK represented the numbers of clock cycles, and the clock period is 0.833 ns (according to DDR4-2400 I/O speed).

To evaluate PIM operations, we use bitmap indices to track users' characteristics.[4] The database stores the gender and the per-day log-in activity of millions of users in memory. We run two queries: how many unique users were active every week for the past w weeks, and how many male users were active each of the past w weeks. To execute these queries, $6w$ bitwise OR, $(2w\text{-}1)$ bitwise AND and $(w + 1)$ bitcount operations need to be performed for all u-users. For all PIM approaches, we accelerate bitwise AND/OR operations using PIM, and bitcount operations are performed on the CPU. We build an in-house simulator to compare performance of the different approaches, where we model latency of PIM operations and data transfer between the CPU and main memory. For all

Published by the IEEE Computer Society

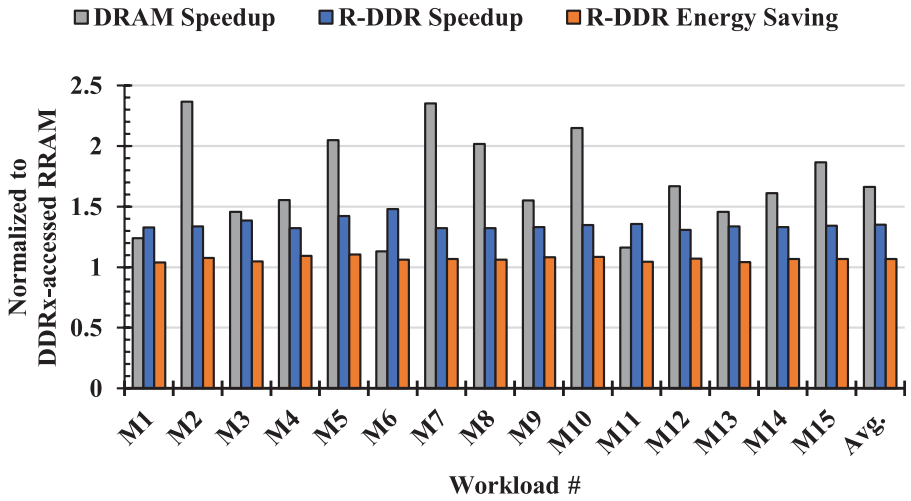

Figure 4. IPC speedup and energy savings of R-DDR normalized to performance and energy consumption of RRAM that is accessed using DDR4-2400 (higher is better). Write intensity (i.e., fraction of write requests in all requests) is also shown for all workloads.

PIM approached, the same I/O frequency and memory organization are used as the baseline (i.e., 4 Gb DDR4-2400). Ambit and Pinatubo numbers are adopted from the original articles and are extended for the configuration used for comparison. Since our workload is data bound, we ignore the time required to compute data on the CPU.

RESULTS

Memory Performance and Energy

Figure 4 compares performance and energy efficiency of three approaches—DDRx-accessed RRAM (baseline); DRAM; and R-DDR accessed RRAM (this work). R-DDR gains 35% speedup and 7% energy savings on average compared to baseline. The reasons for the improved performance include exploitation of partial row-buffer locality, removal of unnecessary steps such as refresh and restore, and redesign of timing constraints similar to DDRx, but specifically for RRAM. The exploitation of partial rowbuffer locality is a major contributor to the superior performance, as it allows us to perform read operations at a much lower latency. In addition, relaxing auxiliary timing constraints such as tWTR and tFAW contributes to the performance.

Energy benefits are thanks to the ability to exploit partial row-buffer locality and reduced execution time. Employing a narrow row-buffer policy reduces the number of reads that must be performed from the RRAM array, hence saving the read energy of requests hitting on the row buffer. Furthermore, reduction in execution time of workload results in lower background energy. However, energy savings of R-DDR is lower than performance improvement compared to baseline because energy consumption is dominated by write requests.

PIM Performance

Figure 5(a) shows the normalized speedups of Pinatubo, Ambit, and the proposed MAGIC-based PIM approach compared to a von Neumann machine for different numbers of weeks and users. For a fair comparison, we include the execution time to compute the end-to-end workload and ignore cycles to send commands from the memory controller to invoke PIM operations since the overhead of sending commands that take a couple of cycles is negligible compared to PIM operation that takes several tens-hundreds of cycles, and it can further be minimized by pipelining PIM execution with sending commands. Note that although MAGIC is proposed previously, the benefits of MAGIC are enabled because of system proposed modifications without which, it is not possible to realize MAGIC. The figure shows that Pinatubo performs similar to the CPU-based approach; Ambit and our system outperform CPU by approximately 5×. Pinatubo's low performance is attributed to its

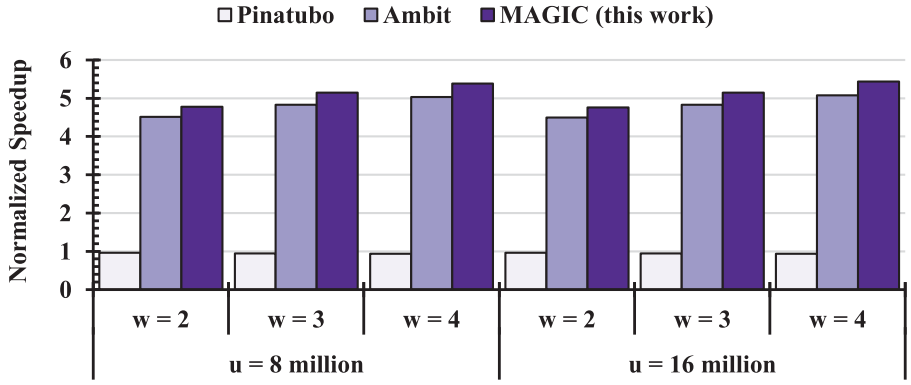

Figure 5. Speedup of Pinatubo, Ambit, and proposed MAGIC-based PIM normalized to von Neumann-based computing machine for query execution in bitmap indices (higher is better). Legends: *w*—#weeks; *u*—#users.

limited compute parallelism due to sense amplifier multiplexing. Both Ambit and MAGIC perform PIM operations in parallel on vectors of size of a full row with similar compute bandwidth. Moreover, since this workload employs $u(w+1)$ bit-read operations to the CPU for bitcount, data transfer between CPU and memory takes around 90% of the execution time in both architectures. Hence, with slightly higher MAGIC latency, MAGIC still performs similar to Ambit. However, benefits of MAGIC approach will be more evident for workloads with greater opportunity to reduce data transfer.

CONCLUSION

RRAM has the potential to solve the DRAM scaling problem and the von Neumann bottleneck. However, merely replacing DRAM with RRAM is not the answer. In this paper, we propose CONCEPT—a column-oriented memory controller for RRAM to optimize its data access pattern as well as exploit PIM capabilities. We show that a memory controller designed around the technological constraints of RRAM can achieve close-to-DRAM performance. Furthermore, supporting PIM operations in RRAM can result in a 5× speedup compared to a state-of-the-art CPU-memory based von Neumann model.

ACKNOWLEDGMENT

This work was carried out when the first author was at the Technion—Israel Institute of Technology. This work was supported in part by the European Research Council under the European Union's Horizon 2020 Research and Innovation Programme under Grant 757259, in part by the Viterbi Fellowship at the Technion Computer Engineering Center, in part by the EU ICT COST Action IC1401, and in part by the Israel Science Foundation under Grant 1514/17.

REFERENCES

1. A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Des. Test*, vol. 39, no. 10, pp. 39–50, Apr. 2017.
2. S. Kvatinsky *et al.*, "MAGIC - Memristor-Aided loGIC," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 61, no. 11, pp. 895–899, Sep. 2014.
3. N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.
4. V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2017, pp. 273–287.
5. S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "PINATUBO: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. 53rd Annu. Des. Automat. Conf.*, Jun. 2016, pp. 1–6.
6. J. Reuben *et al.*, "Memristive logic: A framework for evaluation and comparison," in *Proc. 27th Int. Symp. Power Timing Model., Optim. Simul.*, Sep. 2017, pp. 1–8.
7. X. Cong *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2015, pp. 476–488.
8. H. Y. Lee *et al.*, "Evidence and solution of over-RESET problem for HfOx based resistive memory with sub-ns switching speed and high endurance," in *Proc. Int. Electron Devices Meet.*, Dec. 2010, pp. 19.7.1–19.7.4.

9. J. Zhou, K. H. Kim, and W. Lu, "Crossbar RRAM arrays: Selector device requirements during read operation," *IEEE Trans. Electron Devices*, vol. 61, no. 5, pp. 1369–1376, May 2014.
10. "4Gb one-die DDR4 SDRAM Datasheet Rev. J," Micron, Inc., Boise, ID, USA, Jul. 2017.
11. A. Haj Ali, R. Ben Hur, N. Wald, R. Ronen, and S. Kvatinsky, "Not in name alone: A memristive memory processing unit for real in-memory processing," *IEEE Micro*, vol. 38, no. 5, pp. 13–21, Sep./Oct. 2018.
12. S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM—A general model for voltage controlled memristor," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015.
13. X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.
14. M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Amherst, MA, USA, Sep. 2012, pp. 392–397.

Nishil Talati is currently working toward the PhD degree at the Computer Science and Engineering Department, University of Michigan, Ann Arbor. His current research interests include computer architecture, main memory systems, and emerging memory technologies. He received the B.Eng. degree in electrical engineering from BITS Pilani, India, in 2016, and the MSc degree in electrical engineering from Technion—Israel Institute of Technology in 2018. Contact him at talatin@umich.edu.

Heonjae Ha is currently working toward the PhD degree in electrical engineering at Stanford University. Prior to joining the Ph.D. program, he worked from 2009 to 2013 with SK Hynix, where his main role was to design and validate mobile DRAMs. His current research is focused on understanding and improving the energy efficiency of DRAMs. He received the BEng degree in electronics engineering from Korea University in 2006 and the MS degree in electrical engineering from Stanford University in 2009. Contact him at hunjaeha@stanford.edu.

Ben Perach is currently working toward the PhD degree in electrical engineering at the Technion—Israel Institute of Technology. His current research interests include computer architecture with a focus on processor design, and field-programmable gate arrays, security, and data networks. He received the BSc degree in mathematics from The Hebrew University of Jerusalem, Jerusalem, in 2010, and the MSc degree in electrical engineering from Tel Aviv University in 2017. Contact him at bperach@gmail.com.

Ronny Ronen is a senior researcher at the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology. He was with Intel from 1980 to 2017 in various technical and managerial positions. In his last role, he led the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI). Until 2011, he was a senior staff computer architect in the Intel Development Center in Haifa and before that the director of Microarchitecture Research in that center. In these roles, he led/was involved in the initial definition and pathfinding of major leading-edge Intel processors. Earlier, he led the development of several system software products and tools including the Intel Pentium processor performance simulator and several compiler efforts. He holds more than 70 issued patents and has published more than 20 papers. He received the BSc and MSc degrees in computer science in 1978 and 1979, respectively, both from the Technion—Israel Institute of Technology. He is an IEEE Fellow and was an Intel Senior Principal Engineer. Contact him at ronny.ronen@technion.ac.il.

Shahar Kvatinsky is an assistant professor at the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology. From 2006 to 2009, he was with Intel as a circuit designer and was a Postdoctoral Research Fellow at Stanford University from 2014 to 2015. He is an editor of *Microelectronics Journal* and has been the recipient of the 2015 IEEE Guillemin-Cauer Best Paper Award, the 2015 Best Paper of *Computer Architecture Letters*, Viterbi Fellowship, Jacobs Fellowship, ERC starting grant, the 2017 Pazy Memorial Award, the 2014 and 2017 Hershel Rich Technion Innovation Awards, the 2013 Sanford Kaplan Prize for Creative Management in High Tech, 2010 Benin prize, and six Technion excellence teaching awards. His current research is focused on circuits and architectures with emerging memory technologies and design of energy efficient architectures. He received the BSc degree in computer engineering and applied physics and the MBA degree in 2009 and 2010, respectively, both from the Hebrew University of Jerusalem, and the Ph.D. degree in electrical engineering from the Technion—Israel Institute of Technology in 2014. Contact him at shahar@ee.technion.ac.il.