RC-NVM: Dual-Addressing Non-Volatile Memory Architecture Supporting Both Row and Column Memory Accesses

Shuo Li, Nong Xiao, *Member, IEEE*, Peng Wang, *Member, IEEE*, Guangyu Sun, *Member, IEEE*, Xiaoyang Wang, Yiran Chen, *Fellow, IEEE*, Hai (Helen) Li, *Senior Member, IEEE*, Jason Cong, *Fellow, IEEE*, and Tao Zhang, *Member, IEEE*

Abstract—Although emerging non-volatile memories (NVMs) have been comprehensively studied to design next-generation memory systems, the symmetry of the crossbar structure adopted by most NVMs has not been addressed. In this work, we argue that crossbar-based NVMs can enable dual-addressing memory architecture, i.e., RC-NVM, to support both row- and column-oriented memory accesses for workloads with different access patterns. Through circuit-level analysis, we first prove that such a dual-addressing architecture is only practical with crossbar-based NVMs rather than DRAM. Then, we introduce the RC-NVM architecture from bank, chip and module levels, and propose RC-NVM aware memory controller. We also address the challenges to implement the end-to-end RC-NVM system. Especially, we design a novel protocol to solve the cache synonym problem with very little overhead. Finally, we introduce the deployment of RC-NVM for in-memory databases (IMDBs) and evaluate its performance with IMDBs and well-optimized general matrix multiply (GEMM) workloads. Experimental results show that with only 10 percent area overhead 1) the memory access performance of IMDBs can be improved up to 14.5X, and 2) for GEMM, RC-NVM naturally supports SIMD operations and outperforms the best tiled layout by 19 percent.

Index Terms—Non-volatile memory, crossbar, in-memory database, OLTP, OLAP

1 Introduction

Emerging non-volatile memories (NVMs) have been comprehensively studied to replace or complement DRAM in next-generation memory systems. Prior works mainly exploit the non-volatility, high storage density, and low standby power of NVMs. Unlike DRAM, most NVMs are based on simple two-terminal switching elements, which can be fabricated in a crossbar structure [1]. This structure not only minimizes the feature size of memory cells maximizing the storage density, but also possesses the symmetry property. However, there is still no work to exploit the symmetry of the crossbar structure, which can naturally enable dual-addressing memory architecture to support both row and column memory accesses for workloads with different access patterns, especially hybrid OLTP (on-line transactional processing) and OLAP (on-line analytical processing) in-memory databases (IMDBs). We here take IMDBs as an example to illustrate the need for memory to support both row and column accesses.

An IMDB is a database system that stores a significant part, if not the entirety, of data in main memory to achieve high performance. Compared with traditional disk-based databases, which only buffer small portions of data in main memory, an IMDB primarily relies on main memory for data storage. Conventionally, database workloads are categorized into OLTP and OLAP. OLTP workloads are characterized by a mix of reads and writes to a few rows at a time, which are often latency-critical. On the contrary, OLAP workloads are characterized by bulk sequential scans spanning a few columns, such as computing the sum of a specific column. These two kinds of workloads are usually served by two different types of DBMSs, i.e., transactional processing and data warehouse systems. However, the explicit separation between OLTP and OLAP workloads in IMDBs suffers from 1) the significant waste of the precious memory capacity due to two copies of data resident in memory, and 2) the low data freshness for real-time analysis due to the slow expensive extract-transform-load (ETL) process between the OLTP and OLAP parts [2].

IMDBs with substantial performance improvement have made it possible to process hybrid OLTP and OLAP

- *S. Li is with State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China. E-mail: lishuo12@nudt.edu.cn.*
- *N. Xiao is with School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China. E-mail: xiaon6@sysu.edu.cn.*
- *P. Wang, G. Sun, and X. Wang are with Center for Energy-efficient Computing and Applications, Peking University, Beijing 100080, China. E-mail: {wang_peng, gsun, yaoer}@pku.edu.cn.*
- *Y. Chen and H. Li are with Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708. E-mail: {yiran.chen, hai.li}@duke.edu.*
- *J. Cong is with University of California, Los Angeles, Los Angeles, CA 90095. E-mail: cong@cs.ucla.edu.*
- *T. Zhang is with Pennsylvania State University, State College, PA 16801. E-mail: tao.zhang.0924@gmail.com.*

Manuscript received 23 Apr. 2018; revised 31 July 2018; accepted 27 Aug. 2018. Date of publication 2 Sept. 2018; date of current version 22 Jan. 2019. (Corresponding authors: Nong Xiao and Guangyu Sun.)
Recommended for acceptance by R. F. DeMara.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2868368

workloads (referred to as OLXP [3]) in a single database. However, different workloads require different data access patterns and storage layouts in order to achieve good performance [4]. The row-store layout works well for OLTP workloads, while the column-store layout works well for OLAP workloads. This is because OLTP workloads tend to access most (if not all) of the fields of specific tuples in the database, while OLAP workloads tend to access one or several fields of a large number of tuples. Arulraj et al. [5] proposed a hybrid layout to store hot and cold tuples in a table with the row-/column-store layout respectively. However, data reorganization is needed to dynamically adapt the storage layout, which significantly increases the complexity of the DBMS and degrades the overall performance. Ideally, if the tables stored in memory can be accessed in both row and column directions, IMDBs can naturally support both OLTP and OLAP workloads without data reorganization.

Without any doubt, performance of an IMDB is quite sensitive to the efficiency of accessing data in main memory. Thus, how to optimize memory architecture to facilitate both row-oriented and column-oriented data accesses has become a key instrument in improving its performance. Recently, Seshadri et al. proposed a technique called GS-DRAM [6] to accelerate strided accesses by allowing the memory controller to access multiple values of a strided access pattern from different chips with a single read/write command. However, GS-DRAM is not flexible enough since only power-of-2 strided access patterns can be supported. Considering various tuples in real IMDB tables and kinds of workloads accessing different fields, the benefit of GS-DRAM will be significantly degraded in real IMDB applications where various strided access patterns coexist. In addition, the complexity increase with the number of tables because multiple patterns may exist at the same time.

An ideal solution is to design novel memory architecture to support both row-oriented and column-oriented accesses. Chen et al. proposed dual-addressing DRAM, to which we refer as RC-DRAM throughout the remainder of this work, to enable DRAM to support both row-oriented and column-oriented memory accesses [7]. In RC-DRAM, two transistors and one capacitor are employed to store a single bit of data, in contrast to the common single-transistor-single-capacity DRAM cell. Moreover, one extra pair of wordline and bitline per cell is added to support the column-oriented access. In fact, DRAM cell arrays are the most area-consuming part in DRAM chips. The modifications to DRAM arrays in RC-DRAM lead to unacceptable area overhead, which significantly diminishes the high density advantage of DRAM. Therefore, the design of RC-DRAM is impractical for real applications (see Section 3.4).

As promising candidates to replace or complement DRAM, NVM-based main memory has been proposed to utilized in many fields, including IMDBs [8] and scientific applications [9]. Unlike DRAM, most NVMs are fabricated in a *symmetric crossbar structure*, which can be exploited to easily implement dual-addressing memory to support both row-oriented and column-oriented accesses with negligible area and latency overhead. Based on this observation, we design Row-Column-NVM (RC-NVM) architecture that leverages the symmetry of crossbar-based NVMs to support both row-oriented and column-oriented memory accesses. Contributions of this work are summarized as follows:

- We propose RC-NVM, a novel memory architecture that exploits the symmetric crossbar structure to support both row- and column-oriented accesses. We extend the traditional memory controller to work for RC-NVM.
- We address the challenges in implementing an end-to-end RC-NVM system, including ISA, system and software support. Especially, we propose a novel cache architecture to solve the cache synonym issue with very little overhead.
- We introduce the deployment of RC-NVM for IMDBs and propose group caching technique that combines the IMDB knowledge with the memory architecture to further optimize the system.
- We present the benefits of RC-NVM by evaluating its performance with an OLXP benchmark and a well-optimized general matrix multiply (GEMM) workload. Experimental results show that with only 10 percent area overhead 1) for IMDBs, the memory access performance can be improved up to 14.5X, and 2) for GEMM, RC-NVM naturally supports SIMD operations and outperforms the best tiled layout by 19 percent.

The rest of this paper is organized as follows. We introduce the background and motivation in Section 2. The RC-NVM architecture and end-to-end RC-NVM system design are introduced in Sections 3 and 4. Section 5 discusses the deployment of RC-NVM for hybrid OLTP and OLAP IMDBs. Evaluation methods and results are presented in Section 6. We introduce related work in Section 7, followed by the summary in Section 8.

2 Background and Motivation

We begin with a review of the data layout issues of IMDBs to address the difference between OLTP and OLAP access patterns. Then, we present existing RC-DRAM design and argue that it is impractical due to high area overhead. Finally, we take crossbar-based RRAM as an example to show the potential of RC-NVM design.

2.1 Data Layout Issue of OLXP DBMSs

Relational databases organize data into two-dimensional tables with rows (i.e tuples) and columns (i.e., fields). Although DRAM modules consist of two-dimensional capacitor arrays with wordlines and bitlines, they only support row-orientated accesses. Therefore, from the perspective of operating systems and applications, main memory is a linear address space starting at zero. NVM-based main memories [10], [11] also follow the basic design of single-addressing architecture. The database storage manager must decide how to map the two-dimensional table structures to the linear memory address space.

Most OLTP DBMSs employ the row-store layout. In this case, all fields of a tuple are stored consecutively in memory. Two totally different access patterns from OLTP and OLAP workloads respectively are illustrated in Fig. 1. The upper half shows a typical OLTP transaction that accesses a single tuple, while the lower half demonstrates an OLAP

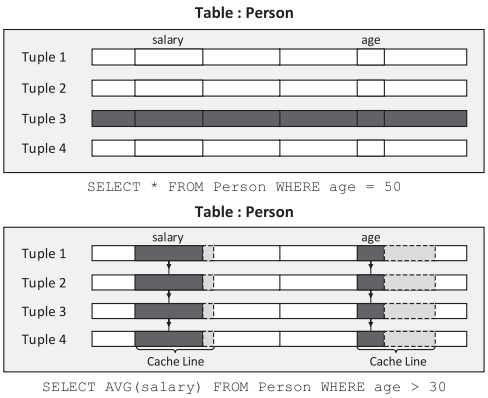


Fig. 1. Access patterns of OLTP & OLAP workloads.

query that scans two fields of all tuples. The row-store layout works well for OLTP workloads, while it is inefficient for OLAP workloads. Since row-store DBMSs unnecessarily access fields that are not needed when executing OLAP queries, this results in the waste of precious cache space and memory bandwidth resources.

An alternative approach, column-store layout, stores a single field of all tuples in a table contiguously. This storage layout works well for column-oriented accesses in OLAP workloads, like performing an aggregate calculation on a specific field. However, it is not ideal for write-heavy OLTP workloads. Currently, the row-store layout is widely used for OLTP DBMSs, while the column-store layout is widely utilized in OLAP scenarios like data warehouses.

Arulraj et al. [5] proposed the hybrid layout, which combines both the row-store and column-store layouts. In essence, hot tuples in a table employ the row-store layout for OLTP transactions, while colder tuples in the same table are stored in the format ideal for OLAP queries. A logical abstraction over this hybrid data layout allows a single execution engine to support hybrid workloads. However, this design increases the complexity of the DBMS and degrades the overall performance due to the additional overhead needed to transfer tuples between two different formats and maintain the consistency of the whole table.

2.2 Crossbar Structure of NVMs

Most of the emerging NVMs are based on simple two-terminal switching elements, such as MRAM, RRAM, PCM and 3D XPoint. To integrate NVM cells into a memory array, there are two type of architectures. The first one is one-transistor and one resistor (1T1R), where each NVM cell is in series with a cell selection transistor, which is used to isolate the selected cell from other unselected cells. The second architecture is the crossbar array, which consists of wordlines and bitlines perpendicular to each other with NVM cells located at the intersections. The crossbar array in principle can achieve $4F^2$ cell area, which has higher integration density than the 1T1R array. Furthermore, the crossbar architecture allows 3D stacking of multiple memory array layers, increasing the effective density further.

Emerging non-volatile memory technologies have been comprehensively studied to implement next-generation persistent memory systems. However, prior works neglected the symmetry of the crossbar structure, which can be exploited to easily implement dual-addressing memories. In this section, we take RRAM as an example to explain why crossbar-based NVMs are feasible for dual-addressing memory design. The similar design can be extended to other crossbar-based NVM technologies, such as MRAM [12], [13], PCM [14] and 3D XPoint [15], [16].

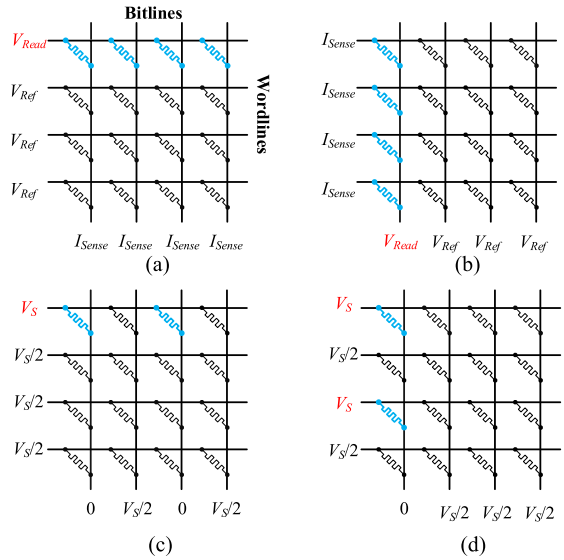


Fig. 2. (a) Read a row in a 4×4 ReRAM crossbar arry, (b) Read a column, (b) SET two cells on a row and (c) SET two cells on a column.

An example of 4×4 crossbar array of RRAM is depicted in Fig. 2a. The crossbar array is consisted of crossing wordlines and bitlines. Each RRAM cell lies at the intersections of wordlines (WLs) and bitlines (BLs). Without access transistors, these cells are directly interconnected to WLs and BLs via top and bottom electrodes. Read and write operations can be performed by activating WLs and BLs with particular voltages.

To read out a row from the array, the target WL will be driven to read voltage V_{Read}. Other WLs are set to the read reference voltage V_{Ref}. By keeping the voltage of BLs being V_{Ref} with current sensing amplifiers, the voltage across the unselected RRAM cells is equal to zero. Thus, data of the target row can be read out by sensing the pass-through current I_{Sense} on each BL as shown in Fig. 2a. Since WLs and BLs are symmetric in such a crossbar array, reading out a column can be realized by simply exchanging operations on WLs and BLs as shown in Fig. 2b.

The write operation requires two steps, RESET and SET. The RESET phase writes "0"s, while the SET phase writes "1"s. Since the only difference between RESET and SET is the polarity of voltages applied across target cells, we here take the SET operation as an example to illustrate row and column write operations in the crossbar array. Fig. 2c shows how to SET two cells on the first row. The target wordline WL_0 is set to V_S, and the corresponding bitlines (i.e., BL_0 and BL_2) are set to *GND*. At the same time, all other WLs and BLs are half biased at $V_S/2$. Therefore, the write voltage V_S is fully applied across the target cells. In this way, the two target cells on WL_0 will be set to "1".

Similarly, to set cells on a column can be easily implemented by control the voltages of wordlines and bitlines. As shown in Fig. 2d, the corresponding wordlines WL_0 and WL_2 are set to V_S, and the target bitline BL_0 is set to *GND*. All other wordlines and bitlines are half biased at $V_S/2$.

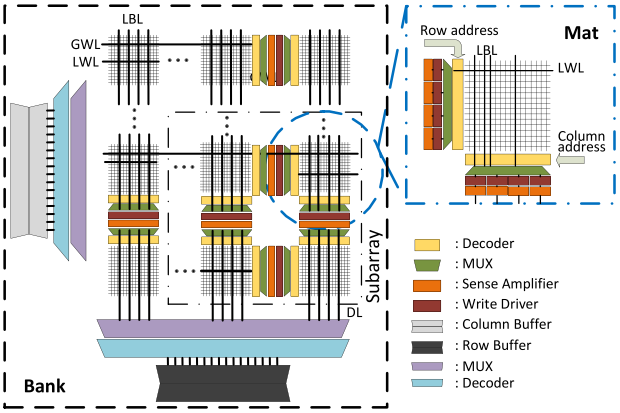


Fig. 3. RC-NVM bank: (a) Schematic view of an RC-NVM bank; (b) Mat organization. (GWL: global wordline; LWL: local wordline; LBL: local bitline; DL: dataline).

A key observation is that there is *no change* required to a crossbar array to enable both row and column accesses because of the symmetry of the RRAM crossbar array. Thus, compared to RC-DRAM, we can implement the RC-NVM design with very little area overhead. Note that RC-NVM is also feasible for crossbar arrays with specific selectors (e.g., FAST selector for RRAM [17] and OTS selector for PCM [18]).

3 The RC-NVM Architecture

We first introduce the physical/logic bank and module architecture of RC-NVM. Especially, we propose peripheral circuitry sharing mechanism to minimize the area overhead. Then, we evaluate the area and latency overhead of RC-NVM. Finally, we introduce the design of RC-NVM aware memory controller.

3.1 RC-NVM Bank

Fig. 3a shows the schematic view of an RC-NVM bank in a chip. Leaving the RRAM array (i.e., mat) unchanged, extra peripheral circuitry is added to support both row-oriented and column-oriented accesses in the same bank. Both WLs and BLs are connected with dedicated decoder, sense amplifier (SA) and write driver (WD) as shown in Fig. 3b. The connection is controlled by multiplexers (MUXs) and control signals from the memory controller. In addition to the existing row buffer, a column buffer is deployed to buffer column-oriented data. A set of adjacent mats in a bank is organized into a single entity called *subarray*. A bank contains multiple subarrays which can provide some parallelism within the same bank, especially in RC-NVM based memory.

Similar to traditional DRAM and NVM, a logic bank consists of a set of banks from every chip in a rank, which are operated in lockstep. Fig. 4 shows a logical abstraction of a logic bank. A logic subarray in the logic bank is the basic access unit of both row-oriented and column-oriented accesses. In RC-NVM, the minimum directionless granularity is 8 bytes, which are exactly the data transfered synonymously on a 64-bit memory bus. Therefore, this 8 bytes comes from 8 chips, each of which provides 8 bits from eight independent mats located in a subarray. Then, a typical row-oriented (column-oriented) 64-byte cache line is composed of eight 8-byte units on a row (column) in a logic subarray as shown in the upper left corner of Fig. 4.

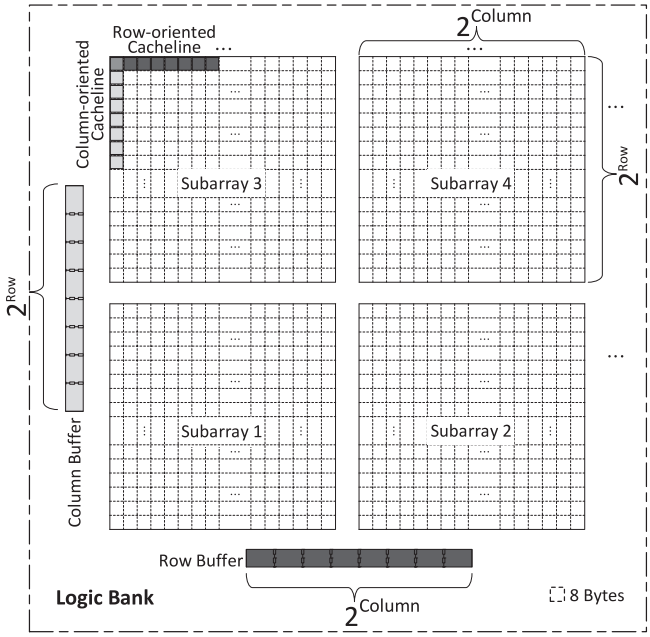


Fig. 4. A logic bank across all 8 chips in a rank.

As we can see in Fig. 4, the data on the intersection of the row and column may be duplicated in both row and column buffer. This data duplication can incur coherence issue if the data is modified in one buffer while the other is not updated. To address this issue, we restrict that the row and column buffer *cannot* be activated for the same subarray simultaneously. If a switch between row and column accesses in a subarray occurs, the memory controller needs to first precharge the active buffer and flush the data back, and then activates the other buffer. In this way, RC-NVM solves the data coherence problem at the cost of some parallelism in the same subarray. However, our experimental results show that the design performs well and the impact is marginal.

We take a row-oriented read operation as an example to explain how addressing is done in RC-NVM. A column read operation has the same flow. Given an access address, the global row decoder does the partial decoding to assert a single global word line (GWL). Then, the local row decoder eventually generates 1-hot signal to assert the local WL (LWL). After that, the column decoder selects the target local bit lines (LBL) and the cells on selected LBL are sensed out. Finally, SAs deliver the data to the row buffer through data lines (DL). Such hierarchical decoding structure can effectively reduce the decoding delay and power consumption when the RC-NVM scales up.

3.2 Sharing of Peripheral Circuitry

In fact, since only a single row/column access is serviced in a subarray at any time, two adjacent mats on a row/column can share the SA and WD between them as shown in the Fig. 3a. This can reduce the area overhead of RC-NVM.

In DRAM, a row buffer is comprised of a row of sense amplifiers, which both sense and buffer data. In NVMs, however, sense amplifiers and latches are kept separately, which allows the decoder and multiplexer to be placed before the row buffer [10], as shown in Fig. 3. This gives an opportunity to share buffers among banks. To further reduce area overhead, adjacent banks in a chip can share

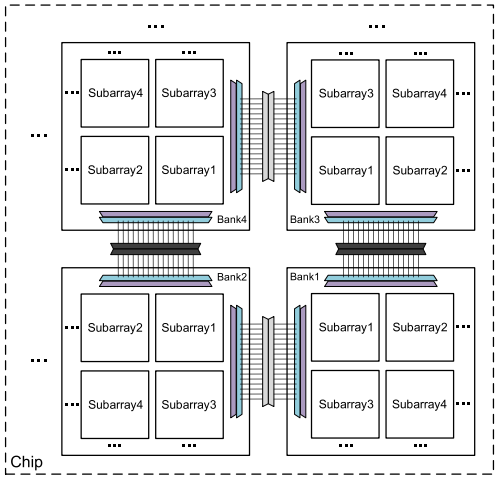


Fig. 5. RC-NVM chip design.

row and column buffers as shown in Fig. 5. Two banks in a row/column compete with each other for the column/row buffer located between them, while two banks in a diagonal have their own row and column buffer, like Bank 1 and Bank4. Considering adjacent data tend to have the same access pattern in terms of orientation, we arrange two banks in a diagonal successively in the address space. With this address mapping scheme and access-pattern-aware memory allocation, the buffer conflict between banks can be minimized to guarantee the bank parallelism.

3.3 RC-NVM Module

The basic overall architecture of an RC-NVM module is similar to a traditional RRAM and DRAM design. It is also organized hierarchically as channel, rank, bank, subarray as shown in Fig. 6.

In this example, there are two ranks on an RC-NVM module. Each rank is composed of eight chips, which work together to form a 64-bit memory bus. In each chip, multiple RC-NVM memory mats are grouped as subarrays to support both row-oriented and column-oriented accesses.

Similar to traditional DRAM modules, the most common error correcting code (ECC), a single-error correction and double-error detection (SECDED) Hamming code can be easily deployed by adding one extra chip per rank, which stores the parity bits. Note that the granularity of ECC is the basic access unit (i.e., 8 bytes) of RC-NVM.

3.4 Area and Latency Overhead

The comparison of area overhead between RC-DRAM and RC-NVM is given in Fig. 7. X-axis shows numbers of WL/BL

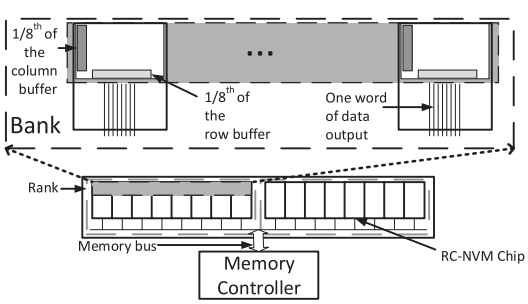


Fig. 6. Overall architecture of an RC-NVM module.

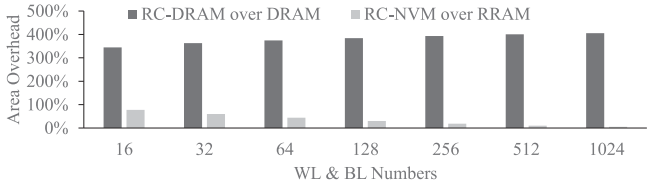


Fig. 7. Area overhead of RC-DRAM and RC-NVM.

in a single memory array, which represents different array sizes. Y-axis shows the area overhead of RC-DRAM and RC-NVM over traditional DRAM and NVM, respectively. The device level parameters of RRAM are from the Panasonic's prototype [19]. The DRAM is modeled based on the parameters from Micron's DDR3 technical specifications [20].

Fig. 7 shows that the RC-DRAM always induces significant area overhead (more than 2X) over the original DRAM. The area overhead is proportional to the number of WL/BL in an array. Therefore, RC-DRAM is not a practical design. On the contrary, the overhead of RC-NVM is much lower. Compared to RRAM, the proposed RC-NVM only requires extra peripheral circuitry while the cell arrays are intact. Thus, the overhead decreases as the size of the arrays increases. As shown in Fig. 7, the overhead drops to 10 percent when the number of WL/BL is 512. Therefore, RC-NVM becomes more attractive with larger array sizes.

Extra peripheral circuitry also induces latency overhead mainly from wire routing. Since more multiplexing transistors are added to the critical path, the read and write latency increase. The latency overhead of multiplexers, however, is trivial because the majority of access latency comes from the cell access and wiring delay. To quantify the latency overhead, we run SPICE simulation and the results are presented in Fig. 8. The latency overhead for RC-NVM is moderate. When the number of WL/BL is 512, the timing overhead is just about 15 percent.

3.5 Memory Requests Scheduling for RC-NVM

To maximize RC-NVM performance, the memory controller, which is responsible for scheduling memory requests, need to be adjusted to exploit extra peripheral circuitry, especially column buffers. To exploit extra column buffers, different subarrays in a bank can be activated to serve row and column requests simultaneously. In addition, the memory scheduler should avoid issuing row and column accesses to the same subarray simultaneously to avoid data synonym problem.

In this section, we extend traditional memory controllers to exploit characteristics of RC-NVM. A memory controller consists of a memory request buffer and a scheduler that chooses the next request to be serviced. Fig. 9 shows the main architecture of a simple but representative memory scheduler (without gray parts).

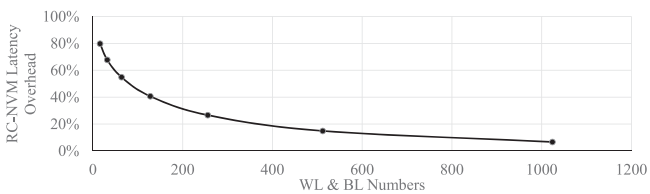


Fig. 8. Latency overhead of RC-NVM.

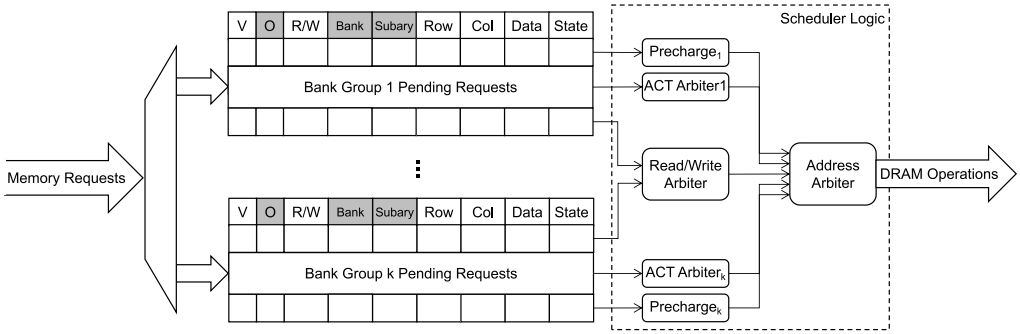


Fig. 9. Memory requests scheduler architecture (the gray parts are extensions for RC-NVM).

As memory requests arrive, they are resolved and buffered in queues with following fields: valid (V), read/write (R/W), address (Row and Col), data, and state (i.e., age of requests) for scheduling policies. Typically, each bank has a separated queue. The scheduler consists of two level arbiters. Each bank has a precharge manager and an activation arbiter. A single read/write arbiter is shared by all banks. The precharge managers, activation arbiters and read/write arbiter send their selected operations to a single address arbiter which grants the shared address resources to one of those operations. These arbiters can adopt different policies to implement different scheduling policies. For common FR-FCFS [21], the address arbiter prioritizes operations from read/write arbiter over others, while the activation arbiters and read/write arbiter all use ordered priority scheduling, in which older requests are given higher priority. The precharge managers use open policy, in which a bank is precharged only if there is no pending reference to the active row and there are pending references to other rows in the bank.

To support RC-NVM requests scheduling, three new fields, orientation (O), bank ID (Bank) and subarray ID (Subary), are added in queues as shown in Fig. 9. The orientation is used to distinguish row and column requests. Instead of one queue per bank, a set of banks (four adjacent banks as shown in Fig. 5) sharing row/column buffers are grouped together to coordinate requests scheduling. Therefore, a bank ID is needed to identify the target bank in the bank group. The target subarray in the bank is recorded by the subarray ID. Similar to traditional bank conflicts, when two requests come to the same subarray, they have to be served sequentially, which is called *subarray conflicts*.

A memory command for a request is ready to schedule only if its scheduling does not violate the timing and resource constraints, including bank and address/data/command bus. Different from the integration of row buffer with a bank in traditional memories, the row/column buffers are decoupled from banks and shared between adjacent banks in RC-NVM. Therefore, when a request is scheduled, its corresponding row/column buffer should be vacant.

Modern memory controllers usually prioritize buffer-hit requests over other requests to exploit buffer locality. For RC-NVM, with the knowledge of both the activated row and column, both row-hit and column-hit requests from different banks can be identified and issued from read/write arbiter.

To simplify the control logic, we treat row and column accesses to the same subarray as regular subarray conflicts. In this way, the data synonym problem in a subarray can be solved by serializing all row- and column-oriented accesses to the same subarray.

4 End-to-End RC-NVM System Design

In this section, we address the challenges to enable both row- and column-oriented accesses for applications. First, two new instructions are introduced to exploit the column-oriented accesses. Second, the memory controller provides separate addressing mappings for each type of access. Third, an application should be aware of the configuration of RC-NVM and explicitly control data layout in RC-NVM. Last but not least, the cache architecture should be modified to work for RC-NVM.

4.1 ISA Support

In order to allow applications to exploit column-oriented accesses, we introduce two new instructions, called *cload* and *cstore*. The details of these two instructions are shown as follows:

```
cloadreg, caddrcstorereg, caddr
```

where `reg` is the destination register, `caddr` is the column-oriented address of the data.

The execution of *cload*/*cstore* follows the same process as traditional *load*/*store*. If a access hits in the on-chip cache, the data is sent to the processor. Otherwise, the access reaches the memory controller. Column-oriented accesses are recognized and resolved by the memory controller, and sent to RC-NVM modules with an additional column-oriented signal. Similar to prior works, this can be implemented by leveraging DDR interface. For example, DDR4 has two reserved address pins, thereby one of them can be used to send this signal [6], [22]. Since traditional row-oriented accesses are still supported by *load* and *store* instructions, traditional applications do not need to be modified.

4.2 Address Mappings

The memory controller needs to support separate addressing mappings for two different accesses. Fig. 10 shows row-/column-oriented address mappings for the same data in RC-NVM. Fig. 10a is a typical row-oriented address for a 32-bit conventional main memory. Fig. 10b demonstrates the corresponding column-oriented address. For the same data (location) in RC-NVM, the only difference is the order of the row bits and column bits in the total 32-bit address. Thus, it is convenient to convert a row-oriented address to its corresponding column-oriented address and vice versa.

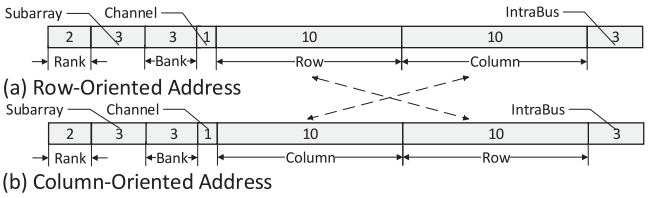


Fig. 10. Address mappings for (a) Row-oriented and (b) Column-oriented accesses.

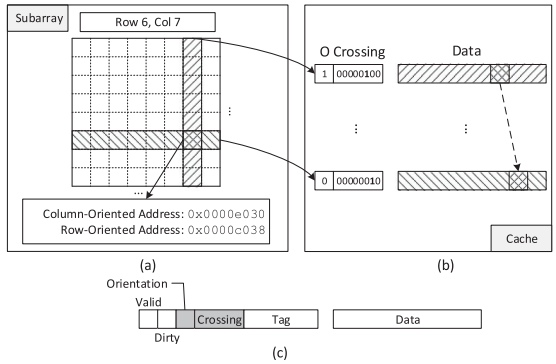


Fig. 11. Illustration of cache architecture for RC-NVM. (a) Data with two addresses in a subarray; (b) Cache blocks in the cache; (c) Extra orientation bit and crossing bits per cache block.

It is easy to find that, when the row-oriented address is increased, the *column bit* is increased. It represents the case of scanning on a physical row. Similarly, for a column-oriented address, increasing the address represents the case of scanning a column.

4.3 Explicit Data Layout Control

To utilize RC-NVM, an application should be aware of the configuration of RC-NVM and explicitly control data layout in RC-NVM, which is similar to traditional databases using raw disks directly and enabling them to manage how data is stored and cached. In order to explicitly control physical data layout in RC-NVM, the application can leverage the *huge-page* technique provided by mainstream operating systems [23], which has been supported in modern processors. By using huge-page, the memory page size is set to 1 GB. Within each huge page, the lower 30 bits of a virtual address and the corresponding physical address are exactly the same. Increasing the memory page size could also reduce the number of TLB misses and improve the performance, which is already used in commercial databases [24].

As discussed in Section 3.1, the basic access unit is a subarray for both row-oriented and column-oriented accesses. Thus, given the address mapping in Fig. 10, the application can explicitly control the data to be accessed in each row-/column-oriented access. Obviously, as long as the subarray bits, which include the *row* and *column bits*, are allocated inside the 30 least significant bits, the application can always explicitly control the data to be accessed. This is practical because the size of a subarray is normally less than 1 GB. Similarly, it also works with the 64-bit memory address. In this work, we use the 32-bit memory address to simplify the discussion.

In real cases, when a computer system with RC-NVM is powered on, the physical geometry information of the equipped RC-NVM, such as the row and column size, is reported to BIOS by the memory controller. The application can access these information with the help of the operating system. Then the data layout can be carefully organized to facilitate the row-/column-oriented accesses.

4.4 Cache Architecture for RC-NVM

In this section, we focus on modifications of the cache architecture to make it work with RC-NVM. First, we introduce how to cache data with two different addresses. Then, we discuss how to solve the data synonym problem in single-core and multi-core scenarios. Finally, we propose elaborated circuitry to check crossing cachelines in parallel.

4.4.1 Caching Data with Dual Addresses

Since data in RC-NVM can be accessed with two different addresses, two copies of every 8 bytes may exist in the cache simultaneously. In order to differentiate these two versions, one extra *orientation bit* per cache line is added as shown in Fig. 11c. When data are loaded into cache with row-oriented addresses, this bit is set to '0', otherwise it is set to '1'.

As shown in the example of Fig. 11a, the 8-byte data can be accessed with `0x0000e030` (column-oriented address) and `0x0000c038` (row-oriented address). When the data are accessed with the column-oriented address, they are loaded into the cache together with other 56-byte data in the same column. As shown in Fig. 11b, the cacheline is placed in the corresponding cache entry and the orientation bit is set to '1'. Similarly, when accessed with the row-oriented address, they are loaded into the cache together with other 56-byte data in the same row. The cacheline is placed in the proper place based on the row-oriented address with the orientation bit set to '0'.

4.4.2 Cache Synonym in a Single-Core Processor

As shown in the previous example, every 8 bytes may have two copies in the cache with row- and column-oriented addresses. This will result in the data synonym problem, which should be resolved to guarantee data consistency. RC-DRAM proposes to solve the problem by separating the cache into two parts and employing a *WURF* cache coherence policy [7]. In this work, instead of partitioning the cache into two parts, we add one extra status bit for each 8 bytes (i.e., the granularity of data synonym) to indicate whether its duplicated copy is cached in the cache. Thus, for a 64-byte cache block, 8 extra status bits are needed, called *crossing bits* as shown in Fig. 11c. Note that each 8-byte granularity lies in two cache blocks at most due to the cacheline alignment of memory accesses.

The basic idea of solving synonym is to keep duplicated data updated at the same time. Extra operations are required for data replacement, write, and write-back operations, which are listed as follows,

- When a cache block is loaded into the cache, the cache controller needs to check all potential cache blocks that may cross with this one whether exist in the cache. For example, in Fig. 11a, a 64-byte row-oriented cache block may be crossed with 8 column-oriented cache blocks in the same logic subarray. Thus, when a row-oriented cache block is loaded into the cache, 8 column-oriented cache blocks are checked.

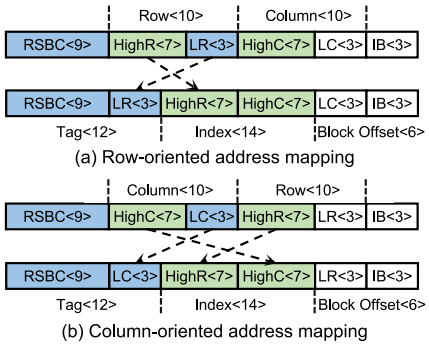


Fig. 12. Physical address mapping in caches.

If any of these 8 column-oriented cache blocks exists in the cache, the crossed region (i.e., 8-byte data) are copied from the column-oriented cache block to the row-oriented cache block so that duplicated data remain the same. At the same time, the corresponding crossing bits are set to '1'. The status of the cache in this example is shown in Fig. 11b.

- When a cache block is written back due to eviction, the crossing bits of its crossed cache blocks are reset to '0'.
- When a cache block is updated in a write operation, if the crossing bit of the modified 8-byte data is equal to '1', the corresponding duplicated data in the crossed cache block are updated at the same time.

4.4.3 Solving Cache Synonym and Coherence Issues

In a multi-core processor, the cache synonym problem also exists as well as the traditional cache coherence issues. These problems can be easily solved by handling these two issues separately in a specific order. The basic rule is: *cache synonym is always solved first, then cache coherence protocols are applied*.

The idea of solving cache synonym problem in a multi-core processor is similar to that in a single-core case. We need to keep duplicated data updated at the same time. Note that the crossing bits are still required. For example, these bits are stored in the cache directory, if a directory based coherence mechanism is employed. Thus, whenever a write operation happens, the crossed cache blocks are updated accordingly.

After that, cache coherence protocols start to work to keep consistent in multiple cores and memory levels. Note that the cache coherence operations only involve the cache blocks in the same address space (either row-oriented or column-oriented). They will not cause further cache synonym problems. Note that there is no change to the existing cache coherence protocols.

4.4.4 Checking Crossing Cachelines

We can find that in the proposed cache synonym solving protocol, there is no extra overhead for a cache read operation. The overhead of a write operation is moderate. Substantial extra overhead is induced in data replacement, since every row-oriented (column-oriented) load from RC-NVM will trigger 8 checks for potential column-oriented (row-oriented) cache blocks. To reduce check overhead, we propose a parallel check mechanism.

In this example, we have a LLC with the following configuration: 8 MB cache size , 64-byte cache block, and 8-way set associativity. It has the same configuration with our evaluation system in Section 6. The 32-bit physical address is divided into two fields: the 6-bit block offset and the 26-bit block address ($2^6 = 64$ and $32 - 6 = 26$). The block address is comprised of index and tag fields. The index can be calculated as follows:

$$2^{\text{Index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}} = \frac{8\text{M}}{64 \times 8} = 2^{14}.$$

Hence, the index is 14 bits wide, and the tag is $26 - 14$ or 12 bits wide. For a row-oriented address shown in Fig. 10a, we assign the 7 most significant bits of the row and column bits to the index, the 9-bit RSBC part (i.e., the combination of Rank, Subarray, Bank and Channel bits) and 3 least significant bits of the row bits to the tag, and 3 least significant bits of the column bits and the 3-bit intra-bus address to the block offset, as shown in Fig. 12. Note that the HighR and HighC parts are swapped in column-oriented address mapping to keep the same order with row-oriented address mapping. With this address mapping scheme, all potential blocks that may cross with a particular cache block will be cached in the same set due to the same index.

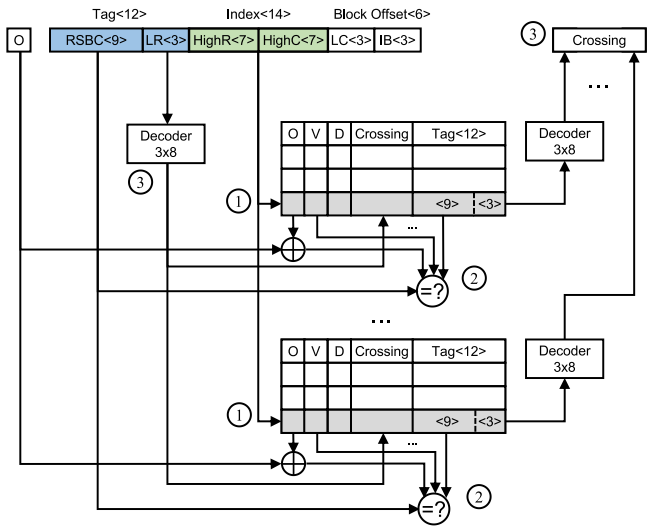


Fig. 13. Checking potential crossed cache blocks.

Given a cache block address A and its orientation O, the checking of all 8 potential cache blocks can be finished in three steps as shown in Fig. 13. First, the 14-bit index is used to locate the target set, labeled as ①. Then, all 8 cache blocks in the set are checked in parallel. Any cache block is identified as a crossed block only if it meets the following three requirements: 1) the *valid* bit is set, 2) the XOR of its *orientation* bit and O is '1', 3) the 9 most significant bits of its tag are equal to the RSBC part of A, labeled as ②. Finally, for crossed blocks, the crossing bits are set based on the decoding results of the 3 least significant bits of the tags, labeled as ③. Note that the circuitry in step 1 and 2 can be partially shared with existing cache lookup operations.

5 RC-NVM Deployment for IMDBs

In this section, we discuss the deployment of RC-NVM for hybrid OLTP and OLAP IMDBs. First, we introduce the placement of IMDB tables in RC-NVM. Then, we present

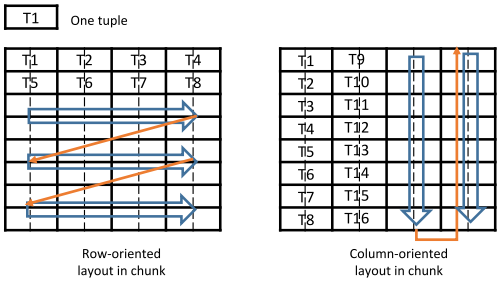


Fig. 14. Two types of data layouts: (a) Row-oriented layout and (b) Column-oriented layout.

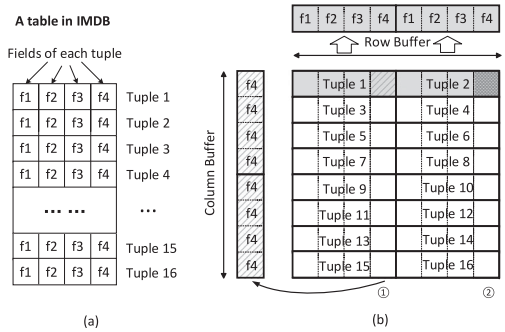


Fig. 15. An example of IMDB table and its layout in RC-NVM.

basic usage of RC-NVM with representative query examples. Finally, we propose group caching technique to solve the problem of wide-field and multi-field accesses.

5.1 IMDB Tables Layout in RC-NVM

As we addressed in Section 4.3, applications can explicitly control physical data layout in RC-NVM to facilitate data accesses. Thus, we can enable more flexible data placement in RC-NVM. The goal is to store IMDB tables efficiently. In this section, we first introduce how to divide a table into small chunks. Then, we discuss how to place these chunks into RC-NVM.

5.1.1 Dividing a Table into Chunks

Since tables in IMDBs are usually very large, we need to divide them into multiple data chunks before placement. This is a common technique in database management systems to store large tables [25]. In this work, a chunk is defined as a rectangle unit of data that can be fit into a subarray of RC-NVM. In other words, a table is divided into chunks when its size is larger than a subarray (i.e., 8 MB in this work) or the tuple size is larger than the row size of the subarray (i.e., 8 KB in this work). After a table is divided, we need to handle intra-chunk and inter-chunk data layout.

5.1.2 Intra-Chunk Data Layout

We present two types of intra-chunk data layouts, i.e., row-oriented layout and column-oriented layout. They are friendly to row-oriented accesses and column-oriented accesses, respectively.

A straightforward row-oriented data layout in a subarray is illustrated in Fig. 14a. Apparently, with such a data layout, tuples in the table are consecutively stored in the row direction. At the same time, their row-oriented addresses are also continuous according to the addressing method in Fig. 10, which is similar to the data layout in traditional main memory. Consequently, it is suitable to traditional IMDBs. With this data layout, row-oriented accesses will achieve the maximum efficiency.

On the other hand, it is easy to understand such a row-oriented data layout is inefficient for column-oriented data accesses, since column-oriented accesses will suffer from more column buffer switchings unless we do not care the access order within each field. To mitigate this problem, we further propose another column-oriented data layout, as shown in Fig. 14b. Tuples are continuously placed in the vertical direction in a subarray. Thus, it is convenient to load the same field of multiple successive tuples with a single column-oriented access. In addition, we will propose a dedicated data access optimization technique for the column-oriented data layout in Section 5.3 to further improve its efficiency for OLXP.

5.1.3 Inter-Chunk Data Layout

At the beginning, all subarrays are empty. Then, tables are created and stored in run-time. Since all IMDB tables have been sliced into chunks, we need to figure out run-time placement policies to fit these chunks in subarrays of RC-NVM. Since both row-oriented and column-oriented accesses are supported, each chunk can be rotated before being placed into a subarray. This is a typical problem of "two-dimensional online bin packing with rotation". Thus, we use the algorithm in Fujita's work to solve this problem [26]. The goal of this algorithm is to minimize the number of subarrays that are used. Please refer to this reference for more details. Note that the placement of IMDB is fully operated in software level (i.e., database memory allocator). It does not require any extra hardware modification.

5.2 Basic Usage of RC-NVM

In this section, we use a simplified example to demonstrate how to leverage both row- and column-oriented accesses in RC-NVM. Fig. 15a illustrates the table used in this example. It is comprised of 16 tuples, each of which consists of four fields. Note that, in order to differentiate a physical row in memory, we use the term "tuple" to represent a row in an IMDB table. To simplify the discussion, the size of all four fields is set to 8 bytes. We assume that this table is stored in a 512-byte RC-NVM subarray in a row-store layout, as illustrated in Fig. 15b. Both the row and column buffer are set to 64 bytes.

Having this table in RC-NVM, we use two SQL queries to illustrate how row- and column-oriented accesses work. The first one is a typical OLTP query as shown in Fig. 16. This query will retrieve all tuples that satisfy the condition (`f3 < '1234'`). Obviously, it is convenient to complete this SQL request with traditional row-oriented accesses. For instance, the first memory request loads two tuples, `T1` and `T2`. Then, the field `f3` in each tuple is read and compared. Finally, data in `T1` and `T2` can be read out accordingly.

The second example for column-oriented accesses is listed in Fig. 17. This typical OLAP query retrieves all `f4` fields and adds them up. If we still use the traditional row-oriented access, all eight memory rows will be loaded sequentially to access field `f4` in each tuple. However, since we have supported column-oriented accesses in RC-NVM, this request is simplified significantly with only two column-oriented memory accesses to read out all fields required.

```
1  for (int i = 1; i <= 16; i++) {
2    if (table->tuple[i].f3 < 1234)
3      Print f1, f2, f3, f4 of tuple[i];
4  }
```

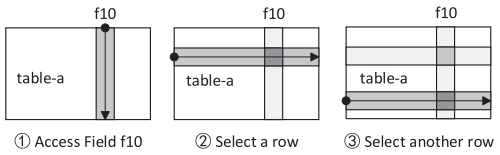
```
SELECT * FROM table WHERE f3 < '1234'
```

Fig. 16. An OLTP SQL example with row-oriented accesses.

```
1  int sum = 0;
2  uint32_t col_addr_1, col_addr_2;
3  col_addr_1 = Row2ColAddr(&(table->tuple[0].f4));
4  col_addr_2 = Row2ColAddr(&(table->tuple[1].f4));
5  for (int i = 1; i <= 8; i++) {
6    uint64_t f4_1 = column_load(col_addr_1);
7    if (f4_1 < 4321)
8      sum += f4_1;
9    col_addr_1 += 8;
10 }
11 for (int i = 1; i <= 8; i++) {
12   uint64_t f4_2 = column_load(col_addr_2);
13   if (f4_2 < 4321)
14     sum += f4_2;
15   col_addr_2 += 8;
16 }
```

```
SELECT SUM(f4) FROM table WHERE f4 < '4321'
```

Fig. 17. An OLAP SQL example with column-oriented accesses.




```
SELECT * FROM table-a WHERE f10 > x
```

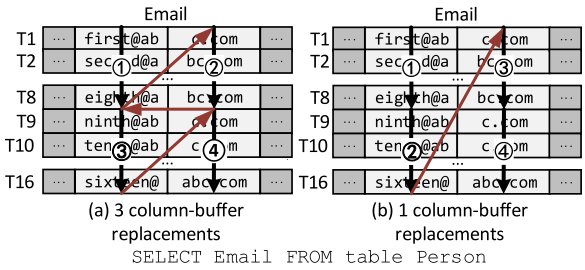
Fig. 18. An SQL example with both row-oriented and column-oriented accesses.

The third example uses both row- and column-oriented accesses as shown in Fig. 18. This query selects certain tuples whose field `f10` is larger than a specific value. Only a few tuples meet the condition. In RC-NVM, we can use column-oriented accesses to scan the `f10` column to check whether the condition is met. If a candidate is found, then the IMDB can issue a row-oriented access to retrieve the tuple. In this case, the data transmitted on the memory bus are all effective, thus the utilization of memory bandwidth is significantly improved.

In the above examples, the size of all fields is 8 bytes which is the same as the column access width. For various non-unit fields, many software optimizations can be exploited. Like the "struct" alignment mechanism in C programming language, IMDB may adjust the order of columns and combine multiple columns with different widths to align on the 8-byte boundary (e.g., 6 + 2, or 36 + 4), or just leave hollows for padding to improve access efficiency. IMDB can also pack them tightly to minimize storage space at the cost of additional column accesses.

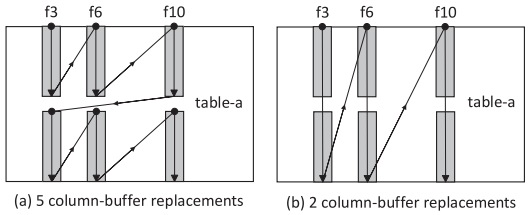
5.3 Group Caching

We observe that the efficiency of column-oriented accesses is degraded when the data are required to be accessed with a specific order. As introduced in Section 3.1, the data width of a column-oriented access in RC-NVM is fixed (i.e., eight bytes). However, the width of fields of an IMDB table vary widely. This situation may degrade efficiency of memory




```
SELECT Email FROM table Person
```

Fig. 19. A wide-field example in column-oriented accesses.




```
SELECT f3, f6, f10 FROM table-a
```

Fig. 20. A multi-field example in column-oriented accesses.

accesses, especially when a field width is larger than the column access width. Such a problem is called the *wide-field* access in this study.

A wide-field access example is given in Fig. 19. In this example, the *Email* field spanning two columns of RC-NVM is indivisible. To get the whole *Email* field, every memory access will trigger a column buffer replacement as shown in Fig. 19a. Although the buffer hit rate is improved in column-oriented accesses shown in Fig. 19b, only half of the field is read out at first, which is meaningless for applications.

Similar to the wide-field access, a multiple-field access example is shown in Fig. 20. This query needs to read a few separate fields in a specific order. In this example, the column-oriented access is also inefficient for the similar reason. Obviously, using column-oriented accesses can traverse each field efficiently *only if* the field order of each tuple is not required as shown in Fig. 20b. However, if the field access order is strictly required, column-oriented accesses are inefficient as shown in Fig. 20a. The reason is straightforward. Each memory access will generate a column-buffer replacement. For instance, three extra column buffer replacements occur in Fig. 20a. Basically, whenever such a Z-style access order is required, the efficiency of column-oriented accesses is degraded. Unfortunately, the row-oriented access is also inefficient due to the waste of the cache space and memory bandwidth.

In order to solve this problem, we propose a novel software-based data caching technique called *group caching*. The basic idea is to cache multiple columns of data as a group in the cache for column-oriented accesses. Then the required data can be accessed in any order with the help of the cache. We modify the query optimizer of the IMDB, which converts SQL statements into memory requests, to generate group caching requests in advance, before the IMDB needs to access a wide field, or several fields. After the data are prefetched into the cache, the IMDB can access the cached data in any order with column-oriented addresses.

One potential issue is unexpected cacheline evictions. The cached data of one thread may be replaced by data retrieved by other threads before they are really accessed, especially in

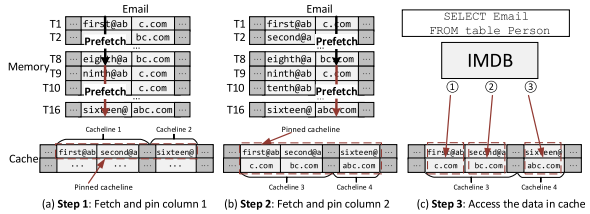


Fig. 21. Illustration of group caching.

TABLE 1
Configuration of Simulated Systems

Processor	4 cores, x86, 2.0 GHz
L1 cache	private, 64B cache line, 8-way associative, 32 KB
L2 cache	private, 64B cache line, 8-way associative, 256 KB
L3 cache	shared, 64B cache line, 8-way associative, 8 MB
Memory controller	32 entry request queues per controller, FR-FCFS
DRAM	DDR3-1333, tCAS: 10, tRCD: 9, tRP: 9, tRAS: 24, Channels: 2, Ranks: 2, Banks: 8, Rows: 65536, Columns: 256, Row buffer size: 2048 B, Capacity: 4 GB, Access time: 14 ns
RRAM	LPDDR3-800, tCAS: 6, tRCD: 10, tRP: 1, tRAS: 0, Channels: 2, Ranks 4, Banks: 8, Rows: 8192, Columns: 1024, Row buffer size: 8192 B, Capacity: 4 GB, Read access time: 25 ns, Write pulse width: 10 ns
RC-NVM	LPDDR3-800, tCAS: 6, tRCD: 12, tRP: 1, tRAS: 0, Channels: 2, Ranks 4, Banks: 8, Rows: 8192, Columns: 1024, Row buffer size: 8192 B, Column buffer size: 8192 B, Capacity: 4 GB, Read access time: 29 ns, Write pulse width: 15 ns, four 512 * 512 mats in a subarray

a multi-core environment. The cache-pinning [27] technique is a solution for this problem. The basic idea is to pin the data in the cache before they are accessed. We still use the wide-field example to demonstrate how this technique works.

As shown in Fig. 21, when the IMDB query planner needs to retrieve a wide field, it will generate column group prefetch requests to read each segment of the wide field. Then the cachelines will be pinned (Step 1 and 2). After the data are used by the IMDB in Step 3, these cachelines will be unpinned. With the help of the group caching, data in a rectangle region can be accessed in either row-oriented or column-oriented way. For different shapes of the target data, the query optimizer can select access methods (row- or column-oriented) to minimize the number of memory accesses.

Apparently, the efficiency of group caching is closely related to the caching size. It is easy to understand that the caching size should not exceed the physical cache size. Due to the fact that group caching will also affect the cache miss rates of other data accesses, the optimal group caching size is not only related to the cache size but also depends on the data access pattern. The performance of wide-field and multi-field accesses with different group caching sizes is evaluated in Fig. 30.

6 Evaluation

In this section, we first introduce the experiment setup and the workloads used for evaluations. Then, we evaluate the performance of RC-NVM and compare it with conventional RRAM and DRAM counterparts with different workloads.

TABLE 2
Benchmark Queries

#	SQL Statement
Q1	`SELECT f3, f4 FROM table-a WHERE f10 > x`
Q2	`SELECT * FROM table-b WHERE f10 > x` (Most of f10 is NOT greater than x)
Q3	`SELECT * FROM table-b WHERE f10 > x` (Most of f10 is greater than x)
Q4	`SELECT SUM(f9) FROM table-a WHERE f10 > x`
Q5	`SELECT SUM(f9) FROM table-b WHERE f10 > x`
Q6	`SELECT AVG(f1) FROM table-a WHERE f10 > x`
Q7	`SELECT AVG(f1) FROM table-b WHERE f10 > x`
Q8	`SELECT table-a.f3, table-b.f4 FROM table-a, table-b WHERE table-a.f1 > table-b.f1 AND table-a.f9 = table-b.f9`
Q9	`SELECT table-a.f3, table-b.f4 FROM table-a, table-b WHERE table-a.f9 = table-b.f9`
Q10	`SELECT f3, f4 FROM table-a WHERE f1 > x AND f9 < y`
Q11	`SELECT f3, f4 FROM table-a WHERE f1 > x AND f2 < y`
Q12	`UPDATE table-b SET f3 = x, f4 = y WHERE f10 = z`
Q13	`UPDATE table-b SET f9 = x WHERE f10 = y`
Q14	`SELECT SUM(f2_wide) FROM table-c` (An OLAP query to read wide field `f2_wide`)
Q15	`SELECT f3, f6, f10 FROM table-a`

6.1 Experiment Setup

We use a cycle-accurate memory simulator, NVMain [28] integrated with gem5 [29] as our system simulator. We simulate an directory based MESI cache coherence protocol. Based on the timing parameters of Panasonic's RRAM model [19], we modified NVMain to quantitatively evaluate the performance of the proposed RC-NVM. We also choose Micron's DRAM [20] as a reference.

The system configuration is listed in Table 1. In the simulated RC-NVM system, we have 2 channels, 4 ranks per channel, 8 banks per rank, and 8 subarrays per bank. Each subarray comprises 1024 rows and 1024 columns, which support both row-oriented and column-oriented memory accesses. The total capacity of the memory system is 4 GB. This configuration exactly matches the address mapping scheme shown in Fig. 10. The well-known FR-FCFS [21] is used as our basic scheduling policy.

6.2 Workloads

As pointed out by prior work, there still lacks standard OLXP benchmarks [5]. Therefore, we developed a synthetic benchmark to represent common enterprise workloads.[1] We first select a number of SQL queries to evaluate performance of RC-NVM. They are typical queries that perform transactional operations (OLTP-style), as well as more complex, read-intensive aggregates on larger sets of data (OLAP-style). These queries composing our benchmark are listed in Table 2. Queries Q14 and Q15 are used to evaluate the effect of the group caching technique. The tuples of table-a and table-b have 16 and 20 fixed length (8-byte) fields respectively, while five variant-length fields in the tuples of table-c, as shown in Fig. 22.

6.3 Micro-benchmark Evaluation

Fig. 23 shows the performance results of RC-NVM, RRAM, and DRAM with eight micro-benchmarks that retrieve the

1. We have open sourced the benchmark at https://github.com/RCNVMBenchmark/RCNVMTrace.

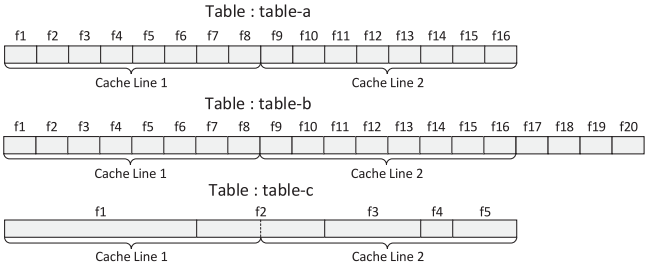


Fig. 22. Three tuple formats in benchmark queries.

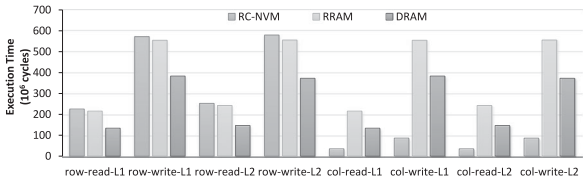


Fig. 23. Micro-benchmark results.

field `f3` and `f10` of all tuples in table-a. The table can be organized as row-oriented layout (labeled as L1) or column-oriented layout (labeled as L2), as shown in Fig. 14. And there are two access directions: row-oriented (labeled as row-read/write) and column-oriented (labeled as col-read/write). For the row-oriented access, to retrieve the field `f3` and `f10` of each tuple needs to load the whole tuple from memory due to alignment restrictions, while the column-oriented access can only get the desired two fields, which reduces the memory access number significantly. For conventional RRAM and DRAM designs, the row-oriented access is used for both directions. For RC-NVM, the row-oriented and column-oriented access are used for different directions accordingly.

From the row-oriented access (the left 4 groups in Fig. 23), we can find that RRAM is 35 percent slower than DRAM, partially because RRAM can only work in a lower operating frequency, as shown in Table 1. And RC-NVM is 4 percent slower than RRAM for the cache coherence overhead. However, RC-NVM outperforms RRAM and DRAM when the IMDB table is accessed in the column-oriented direction. The execution time is reduced by 76 percent in the row-oriented layout (L1) and 77 percent in the column-oriented layout (L2) compared to DRAM. This demonstrates the advantage of column-oriented accesses supported by RC-NVM. Since RC-NVM performs better with the column-oriented layout, we will choose the column-oriented layout as the default to maximize the performance of RC-NVM in the following experiments.

6.4 Queries Evaluation

Fig. 24 presents the execution time of the SQL-query benchmark set consisting of queries Q1 to Q13. Compared with original RRAM and DRAM, the execution time of these benchmark queries on RC-NVM is reduced by 71 and 67 percent on average, respectively. All these results show similar trends, i.e., the performance of RC-NVM is better than DRAM, and DRAM is faster than RRAM. There is only one exception for query Q3, since Q3 is translated into sequential row-oriented memory accesses, which are suitable for DRAM. Compared to RRAM and DRAM, the performance of IMDB can be improved up to 14.5X and 13.3X in the best case (Q6), respectively. Compared with GS-DRAM [6], the performance is improved by 2.37x on average. The reason of performance improvement with RC-NVM can be explained by a combination of three factors.

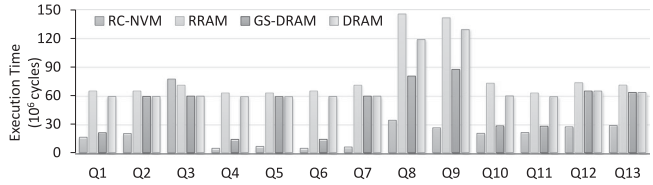


Fig. 24. SQL benchmark results.

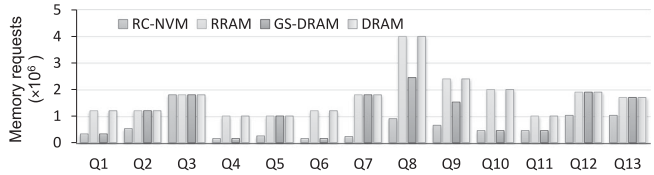


Fig. 25. Numbers of memory accesses.

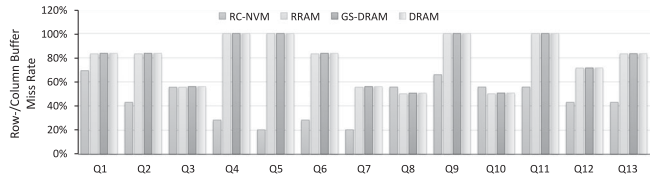


Fig. 26. Comparison of row/column buffer miss rates.

First, by using both row-oriented and column-oriented accesses, the total number of memory requests can be greatly reduced. As shown in Fig. 25, the numbers of memory accesses of RRAM and DRAM in all 13 queries are the same since they can only use traditional row-oriented memory accesses. However, memory access numbers are greatly reduced in RC-NVM, even considering the effect of the cache synonym problem. The number of memory accesses of RC-NVM is less than a third of those of DRAM/RRAM on average. In other words, the IMDB on RC-NVM has two alternative ways to access data, then it can select the best combination of access methods to effectively utilize the precious memory bus resource. For GS-DRAM, memory accesses are only reduced for several queries with power-of-2 strided accesses, like Q1, Q4, and Q6. For queries Q2, Q3 and Q5, GS-DRAM cannot work. Thus, it shows no improvement over conventional DRAM.

Second, the decrease of row/column buffer miss rates also contributes to the performance improvement. In RC-NVM, IMDB has greater possibility to avoid row/column buffer misses caused by strided accesses. Fig. 26 shows RC-NVM achieves a 38 percent decline in the buffer miss rate, which comprising both row and column buffer misses. The buffer miss rates are not reduced after using GS-DRAM.

Third, from Fig. 27 we can see that the extra overhead to solve the cache synonym and coherence of RC-NVM lies in the range of 0.2 to 3.4 percent. On average, the cache coherence overhead introduced by RC-NVM is only 1.06 percent, which is negligible. Note that we do not count the extra coherence overhead for GS-DRAM in experiments since the details are not clearly described in the paper [6].

In previous experiments, we use the RRAM model from Panasonic's RRAM model [19]. In order to reflect the impact

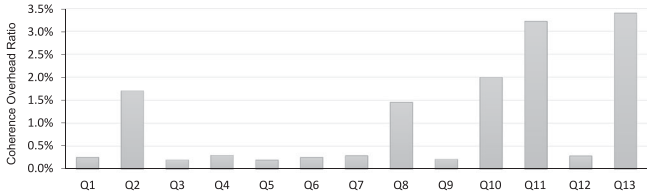


Fig. 27. Cache synonym and coherence overhead.

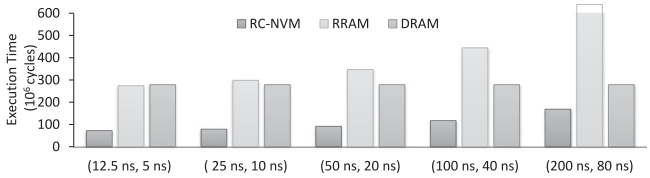


Fig. 28. RC-NVM read latency sensitivity results.

of different RRAM technologies on efficiency of RC-NVM, we perform a sensitivity analysis. As shown in Fig. 28, we scale the read and write latency to different values and compare the average execution time results. We can find that RC-NVM can still outperform DRAM even when the read and write latency are in the level of several hundreds of cycles.

6.5 Effect of RC-NVM Aware Scheduling

Fig. 29 shows the performance of RC-NVM aware scheduling and FR-FCFS which can identify both row and column buffer hits. For queries accessing two tables (Q8 and Q9), RC-NVM aware scheduling can achieve 22.5 percent performance improvement compared to FR-FCFS. In our evaluation, the two tables are stored in different subarrays of a series of banks. For FR-FCFS, accesses to data of two tables stored in the same bank must be serialized due to bank conflicts, while RC-NVM aware scheduling can exploit subarray-level parallelism in the same bank by accessing two tables simultaneously for orthogonal accesses in terms of orientation. For others queries without subarray-level parallelism, they have similar performance because both of them can identify row/column buffer hits and exploit channel and bank level parallelism.

6.6 Effect of Group Caching

By applying group caching optimization, RC-NVM can further achieve performance improvement with relatively small last-level cache usage. The effects of this optimization are shown in Fig. 30. The numbers in the legend indicate how many cachelines are filled at one time. It is apparent that larger group caching sizes achieve better performance. For example, we can achieve a 15 percent performance improvement when the group caching size is set to 128 cachelines for each column.

6.7 Energy Consumption

We also evaluate the energy consumption of different memory configurations. As shown in Fig. 31, due to high static power consumption, the energy consumption of DRAM is 4.5X more than RRAM's. For queries that GS-DRAM can work, like Q1, Q4 and Q6, GS-DRAM consumes much less energy than DRAM. The energy is mainly determined by the total execution time and the number of memory accesses. Due to much less memory requests, RC-NVM consumes much

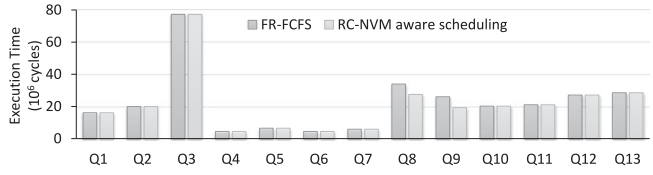


Fig. 29. Impact of RC-NVM aware scheduling.

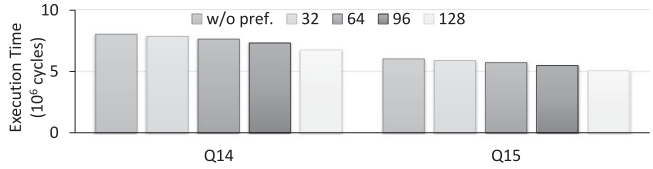


Fig. 30. Impact of Group Caching optimization.

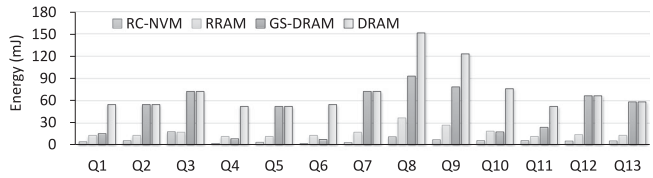


Fig. 31. Energy consumption results.

less energy than RRAM for most queries except for Q3. On average, RC-NVM reduces energy by 66.2 percent compared to RRAM. Since the dynamic power of write/read operations dominates the energy consumption of RRAM, the energy consumption of extra peripheral circuitry in RC-NVM can be covered by the energy reduction of less memory requests.

6.8 Scientific Computing: GEMM

RC-NVM can also give performance improvement for General Matrix Multiply, which is an important kernel in many scientific computations. When two matrices are multiplied, one is accessed in the row-major order, while the other is accessed in the column-major order. Since a matrix may be accessed in both row and column directions, the same data layout dilemma also exists in GEMM.

There are two main optimization methods for GEMM on CPU. The first one is tiling, in which smaller blocks of the whole matrix are fetched into the cache and using them to the most before evicted to amortize the cost of fetching data over useful computations. The second one is single instruction multi-data (SIMD), in which a small batch of data will be processed every time. Although multiple float point multiplications and additions can be processed simultaneously, multiple data fetches are still needed to gather values from different cachelines. To minimize cache misses, a simple data reorganization process called *packing* can be exploited, which reorders the elements of matrix blocks based on the memory access pattern of the multiplication [30]. On one hand, RC-NVM can support both row-oriented and column-oriented accesses to the same matrix. As a result, RC-NVM can directly load target data jnto SIMD registers, eliminating the packing process. On the other hand, since the packing is a data-intensive process with low computing complexity, it is suitable to be offloaded to memory to execute using processing-in-memory (PIM) [31].

Fig. 32 shows the performance of GEMM with different mechanisms. DRAM/RRAM-tiled mechanisms exploit the

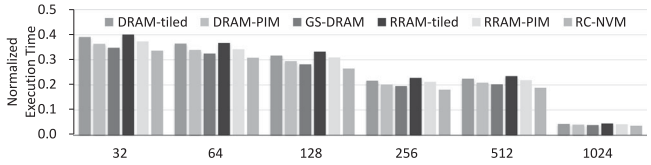


Fig. 32. GEMM performace normalized to the non-tiled baseline.

conventional tiling and packing method. DRAM/RRAM-PIM utilize the PIM accelerator proposed in [31] to implement packing. All results are normalized to a non-tiled version for different matrix sizes. RC-NVM improves the performance of GEMM by 19 percent on average compared to the best tiled version on RRAM. Note that RC-NVM outperforms GS-DRAM by 6 percent due to its high efficiency as explained in Section 6.4. Moreover, RC-NVM and GS-DRAM outperform the PIM accelerator by 9.3 and 3.3 percent respectively. This is because RC-NVM and GS-DRAM naturally enables both row and column memory accesses for GEMM, eliminating the packing process totally.

Emerging 3D-stacked DRAM architectures enable PIM or near-memory processing (NMP) to reduce the data movement between memory and the CPU by offloading part data-intensive task to memory. However, due to the limited compute capability of PIM processing logic, offloaded parts are usually simple functions requiring relative little and simple operations such as basic arithmetic and bitwise operations [31], [32]. Different from PIM, RC-NVM enables NVM-based memory to support both row and column accesses, which can improve the performance of different kinds of workloads with hybrid access patterns. Moreover, as promising replacement for DRAM, we believe emerging NVM techniques can also take advantage of PIM techniques. Therefore, PIM and RC-NVM can be combined to improve the memory performance. The data processed with PIM logic can also benefit from the dual-addressing memory architecture, since the subarray in banks is the basic unit to support both row and column accesses.

7 Related Work

In this section, we discuss prior works that aim to enhance the performance of the memory system, and improve the efficiency of OLTP and OLAP queries in IMDBs.

High Performance Memory Architectures. Many previous works introduce new memory architectures for either achieving lower latency or higher parallelism. SALP [33] exploits the subarray-level parallelism to mitigate the performance impact of bank conflicts in DRAM. A memory scheduling scheme is proposed for Non-Volatile Dual In-line Memory Module (NVDIMM) to minimize the interference between the native and I/O-derived memory traffic [34]. Our mechanisms are orthogonal to these works, and can be applied together with them to further increase memory system performance. Dual-addressing memory [7] (RC-DRAM) enables DRAM to support both row and column memory accesses. GS-DRAM improves the performance of power-of-2 strided memory accesses by changing the organization and access mechanism of traditional DRAM modules [6]. Compared to them, RC-NVM enables crossbar based NVM to efficiently perform both row and column accesses with considerable flexibility and small overhead.

In-Memory Database Optimizations. Various workload characterization studies provide detailed analysis of the time breakdown for databases running on a modern processor, and reveal that databases suffer from high memory-related processor stalls. This is caused by a huge amount of data cache misses [35], which account for 50-70 percent for OLTP workloads [36] to 90 percent for DSS workloads [37], of the total memory-related stall.

Data layouts have a considerable influence on the memory utilization and performance of in-memory databases. To utilize the memory more efficiently, some work re-organizes the records in a column store [38], [39]. Columnar layout favors OLAP workload such as scan-like queries, which typically only needs a few columns of relational table. This layout can achieve good cache locality [40], and can achieve better data compression [41], but has a negative impact for OLTP queries that need to operate on the row level [38], [41], [42].

Some IMDBs try to support OLXP using software methods. There have been several attempts to build databases by means of a hybrid of row and column layouts. For example, SAP HANA [41] supports both row- and column-oriented physical representations of relational tables, in order to optimize different query workloads. It organizes data layout for both efficient OLAP and OLTP with multilayer stores consisting of several delta row/column stores and a main column store, which are merged periodically. Arulraj et al. propose a continuous reorganization technique to shape table's physical layout in either row-stores or column-stores [5]. However, the row-column transformation involves significant data copying overhead and does not work in fully interleaved OLXP workload. In addition, none of them has direct support of memory hardware, which is not enough for performance-critical applications using IMDB.

8 Conclusion

We exploit the symmetry of the crossbar structure adopted by most NVMs to implement dual-addressing memory architecture with very little area and latency overhead. This novel architecture can support row-/column-oriented memory accesses for workloads with different access patterns. With a minor extension to the ISA and the help of huge-page technique, applications can explicitly control the data layout in RC-NVM and issue proper memory accesses to efficiently utilize the precious cache capacity and memory bandwidth. After using RC-NVM architecture, we can achieve even better performance than the DRAM counterpart, although NVM device has a lower access speed. To this end, RC-NVM is considered to be an attractive solution to provide both large capacity and high performance.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No.61433019, U1435217, 61572045) and National Science Foundation (No.1744111S, 1725456).

References

[1] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electron.*, vol. 125, pp. 25–38, 2016.

[2] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 195–206.
[3] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Frber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner, "Towards scalable real-time analytics: An architecture for scale-out of OLxP workloads," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1716–1727, 2015.
[4] I. Alagiannis, S. Idreos, and A. Ailamaki, "H2O: A hands-free adaptive store," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1103–1114.
[5] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 583–598.
[6] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses," in *Proc. 48th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2015, pp. 267–280.
[7] Y. H. Chen and Y. Y. Liu, "Dual-addressing memory architecture for two-dimensional memory access patterns," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2013, pp. 71–76.
[8] J. Arulraj and A. Pavlo, "How to build a non-volatile memory database management system," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1753–1758.
[9] D. Li, J. S. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu, "Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 945–956.
[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 2–13.
[11] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 476–488.
[12] R. Dorrance, J. G. Alzate, S. S. Cherepov, P. Upadhyaya, I. N. Krivorotov, J. A. Katine, J. Langer, K. L. Wang, P. K. Amiri, and D. Markovic, "Diode-MTJ crossbar memory cell using voltage-induced unipolar switching for high-density MRAM," *IEEE Electron Device Lett.*, vol. 34, no. 6, pp. 753–755, Jun. 2013.
[13] M. Wang, W. Cai, K. Cao, J. Zhou, J. Wrona, S. Peng, H. Yang, J. Wei, W. Kang, Y. Zhang, J. Langer, B. Ocker, A. Fert, and W. Zhao, "Current-induced magnetization switching in atom-thick tungsten engineered perpendicular magnetic tunnel junctions with large tunnel magnetoresistance," *Nature Commun.*, vol. 9, no. 1, 2018, Art. no. 671.
[14] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, pp. 143–143, Jan./Feb. 2010.
[15] Intel and Micro, "Intel and Micron produce breakthrough memory technology," (2018). [Online]. Available: https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology
[16] Intel, "Reimagining the data center memory and storage hierarchy," (2018). [Online]. Available: https://newsroom.intel.com/editorials/re-architecting-data-center-memor y-storage-hierarchy
[17] S. H. Jo, T. Kumar, S. Narayanan, W. D. Lu, and H. Nazarian, "3D-stackable crossbar resistive memory based on Field Assisted Superlinear Threshold (FAST) selector," in *Proc. IEEE Int. Electron Devices Meeting*, 2014, pp. 6.7.1–6.7.4.
[18] S. R. Ovshinsky, "Reversible electrical switching phenomena in disordered structures," *Phys. Rev. Lett.*, vol. 21, pp. 1450–1453, 1968.
[19] A. Kawahara, R. Azuma, Y. Ikeda, K. Kawai, Y. Katoh, K. Tanabe, T. Nakamura, Y. Sumimoto, N. Yamada, N. Nakai, S. Sakamoto, Y. Hayakawa, K. Tsuji, S. Yoneda, A. Himeno, K. I. Origasa, K. Shimakawa, T. Takagi, T. Mikawa, and K. Aono, "An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2012, pp. 432–434.
[20] Micron, "DDR3 SDRAM," (2018). [Online]. Available: https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/ 4gb_ddr3_sdram.pdf
[21] S. Rixner, "Memory controller optimizations for web servers," in *Proc. 37th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2004, pp. 355–366.
[22] DDR4 SDRAM STANDARD. (2018). [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd79–4a
[23] Hugepages, (2018). [Online]. Available: https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt
[24] Configuring hugepages for oracle database. (2018). [Online]. Available: https://docs.oracle.com/cd/E37670_01/E37355/html/ol_config_hugepages.ht ml
[25] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proc. EDBT/ICDT Workshop Array Databases*, 2011, pp. 36–47.
[26] S. Fujita and T. Hada, "Two-dimensional on-line bin packing problem with rotatable items," *Theoretical Comput. Sci.*, vol. 289, no. 2, pp. 939–952, 2002.
[27] F. Zyulkyarov, N. Hyuseinova, Q. Cai, B. Cuesta, S. Ozdemir, and M. Nicolaides, "Method for pinning data in large cache in multi-level memory system," U.S. Patent 9 645 942, 2017.
[28] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul.–Dec. 2015.
[29] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[30] Google, "gemmlowp: A small self-contained low-precision GEMM library," (2018). [Online]. Available: https://github.com/google/gemmlowp
[31] A. Boroumand, P. Ranganathan, O. Mutlu, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, and A. Knies, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2018, pp. 316–331.
[32] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented programmable processing in memory," in *Proc. 23rd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 85–98.
[33] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012, pp. 368–379.
[34] R. Chent, Z. Shao, and T. Li, "Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.
[35] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh, "An analysis of database workload performance on simultaneous multithreaded processors," in *Proc. 25th Annu. Int. Symp. Comput. Archit.*, 1998, pp. 39–50.
[36] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker, "Performance characterization of a quad Pentium Pro SMP using OLTP workloads," in *Proc. 25th Annu. Int. Symp. Comput. Archit.*, 1998, pp. 15–26.
[37] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a modern processor: Where does time go?" in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 266–277.
[38] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores versus row-stores: How different are they really?" in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 967–980.
[39] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1664–1665, 2009.
[40] H. Plattner, "A common database approach for OLTP and OLAP using an in-memory column database," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 1–2.
[41] V. Sikka, F. Frber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhvd, "Efficient transaction processing in SAP HANA database: The end of a column store myth," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 731–742.
[42] M. Grund, J. Krger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "HYRISE: A main memory hybrid storage engine," *Proc. VLDB Endowment*, vol. 4, no. 2, pp. 105–116, 2010.

Shuo Li received the master's degree in computer science from the NUDT, China, in 2012. He is working toward the PhD degree in computer science at the National University of Defense Technology (NUDT), China. His interest includes computer architecture and non-volatile memory technology.

Nong Xiao received the BS, MS and PhD degrees of computer science from the NUDT, China. Now he is a professor in School of Data and Computer Science, Sun Yat-sen University. His current research interest includes large-scale storage system, network computing, and computer architecture. He is a member of the IEEE.

Peng Wang received BS and PhD degrees from Peking University, Beijing, China, in 2010 and 2017, respectively. He is currently a senior engineer with Huawei Technologies, working on storage-system design and implementation. His research interests include storage system and computer architecture. He is a member of the IEEE.

Guangyu Sun received the BS and MS degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively, and the PhD degree in computer science from Pennsylvania State University, State College, Pennsylvania, in 2011. He is currently an associate professor of CECA at Peking University, Beijing, China. His research interests include computer architecture, VLSI design, and electronic design automation (EDA). He has published more than 60 journals and refereed conference papers in these areas. He has also served as a peer reviewer and technical referee for several journals, which include the *IEEE Micro*, the *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, etc. He is a member of the CCF and IEEE.

Xiaoyang Wang received the BS degree in computer science from Peking University, China, in 2015, where he is currently working toward the PhD degree in the School of Electrical Engineering and Computer Science. His current research interests include distributed systems and high-performance storage systems.

Yiran Chen received the BS and MS degrees (both with honor) from Tsinghua University and the PhD degree from Purdue University, in 2005. After five years in industry, he joined University of Pittsburgh, in 2010 as assistant professor and then promoted to associate professor with tenure, in 2014, held Bicentennial Alumni Faculty fellow. He now is an tenured associate professor of the Department of Electrical and Computer Engineering, Duke University and serving as the co-director of Duke Center for Evolutionary Intelligence (CEI). He is an associate editor of the *IEEE Transactions on Neural Networks and Learning Systems*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *IEEE Design & Test of Computers*, the *IEEE Embedded Systems Letters*, the *ACM Journal on Emerging Technologies in Computing Systems*, the *ACM Transactions on Cyber-Physical Systems*, and served on the technical and organization committees of more than 40 international conferences. He received 5 best paper awards and 15 best paper nominations from international conferences. He is the recipient of NSF CAREER award and ACM SIGDA outstanding new faculty award. He is also an IEEE fellow.

Hai (Helen) Li (M08-SM16) received the BS and MS degrees from Tsinghua University, Beijing, China, and the PhD degree from the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana. She currently is the Clare Boothe Luce associate professor with the Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina. Prior to it, she was with Qualcomm, Intel, Segate, the Polytechnic Institute of New York University and the University of Pittsburgh. She serves as associate editor of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *IEEE Transactions on Multi-Scale Computing Systems*, the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, the *IEEE Transactions on Embedded Computing Systems*, the *IEEE Consumer Electronics Magazine*, the *ACM Transactions on Design Automation of Electronic Systems* and *IET Cyber-Physical Systems: Theory & Applications*. She was the general chair of ISVLSI, ICCE, ISQED and GLSVLSI, and the technical program chair of SoCC, iNIS, GLSVLSI. She also served on the ACM/SIGDA Outstanding PhD Dissertation Award Selection Committee, the Program chair for ACM SIGDA summer school (DASS), the Executive Committee of ISVLSI, GLSVLSI and iNIS, and the Technical Program Committee members of more than 20 international conference series. She received the NSF Faculty Early Career Development Award (CAREER) in 2012, the DARPA Young Faculty Award (YFA) in 2013, TUM-IAS Hans Fisher fellowship, Technische Universitt Mnchen, Germany in 2017. She is a senior member of IEEE and a distinguished member of ACM.

Jason Cong received the BS degree in computer science from Peking University, in 1985, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign, in 1987 and 1990, respectively. Currently, he is a Chancellor's professor with the Computer Science Department, also with joint appointment from the Electrical Engineering Department, University of California, Los Angeles, the director of Center for Domain-Specific Computing (CDSC), and the director of VLSI Architecture, Synthesis, and Technology (VAST) Laboratory. He served as the chair the UCLA Computer Science Department from 2005 to 2008, and is also a distinguished visiting professor at Peking University. His research interests include synthesis of VLSI circuits and systems, programmable systems, novel computer architectures, nano-systems, and highly scalable algorithms. He has more than 400 publications in these areas, including 10 best paper awards, two 10-Year Most Influential Paper Awards. He was elected to an IEEE fellow in 2000 and ACM fellow in 2008, and received two IEEE Technical Achievement Awards, one from the Circuits and System Society (2010) and the other from the Computer Society (2016).

Tao Zhang received the BS and MS degrees from Peking University, and the PhD degree from Pennsylvania State University, in 2014. He is now working as a SoC performance analysis engineer in Apple Corporation. His research interests mainly fall into computer architecture, memory subsystem design, and 3D IC. He has published 21 papers in various top-class conferences. He also serves as a peer reviewer for journals and conferences in the fields of computer architecture, VLSI design, and EDA. He is a member of ACM and IEEE.