RC-NVM: Enabling Symmetric Row and Column Memory Accesses for In-Memory Databases

Peng Wang[1], Shuo Li[2], Guangyu Sun[1], Xiaoyang Wang[1], Yiran Chen[3], Hai (Helen) Li[3], Jason Cong[4], Nong Xiao[2,5], and Tao Zhang[6]

[1]Center for Energy-efficient Computing and Applications, Peking University, China
[2]State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, China
[3]Department of Electrical and Computer Engineering, Duke University, USA
[4]University of California, Los Angeles, USA
[5]School of Data and Computer Science, Sun Yat-sen University, China
[6]Pennsylvania State University, USA
{wang_peng, gsun, yaoer}@pku.edu.cn *{lishuo12, nongxiao}@nudt.edu.cn*
{yiran.chen, hai.li}@duke.edu *cong@cs.ucla.edu* *tao.zhang.0924@gmail.com*

ABSTRACT

Ever increasing DRAM capacity has fostered the development of in-memory databases (IMDB). The massive performance improvements provided by IMDBs have enabled transactions and analytics on the same database. In other words, the integration of OLTP (on-line transactional processing) and OLAP (on-line analytical processing) systems is becoming a general trend. However, conventional DRAM-based main memory is optimized for row-oriented accesses generated by OLTP workloads in row-based databases. OLAP queries scanning on specified columns cause so-called *strided* accesses and result in poor memory performance. Since memory access latency dominates in IMDB processing time, it can degrade overall performance significantly.

To overcome this problem, we propose a dual-addressable memory architecture based on non-volatile memory, called RC-NVM, to support both row-oriented and column-oriented accesses. We first present circuit-level analysis to prove that such a dual-addressable architecture is only practical with RC-NVM rather than DRAM technology. Then, we rethink the addressing schemes, data layouts, cache synonym, and coherence issues of RC-NVM in architectural level to make it applicable for IMDBs. Finally, we propose a group caching technique that combines the IMDB knowledge with the memory architecture to further optimize the system. Experimental results show that the memory access performance can be improved up to 14.5X with only 15% area overhead.

1. INTRODUCTION

Relational database systems have been the backbone of enterprise applications for more than 30 years. However, traditional disk-based databases fail to satisfy the needs of ultra-low latency service and real-time data analytics as a consequence of high I/O access latency. For instance, high frequency trading applications must issue orders in several microseconds, which is impossible to achieve using disk-based databases.

With increasing capacity and dropping price of DRAM modules, memory systems capable of storing huge amounts of data have become affordable. An in-memory database (IMDB) is a database system that keeps a significant part, if not the entirety, of data in main memory to achieve high query 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. Since an IMDB almost eliminates the I/O bottleneck between a fast RAM and a slow disk, a considerable performance gain can be expected. In recent years, interests in IMDBs are booming in both academic and industrial areas. Examples of well-known IMDB research projects include MonetDB [1], H-Store/VoltDB [2, 3], HyPer [4], and Hyrise [5]. There are also many commercial systems published by database vendors, such as Oracle TimesTen [6], IBM SolidDB [7], Microsoft Hekaton [8], and SAP HANA [9].

Conventionally, database workloads are categorized into OLTP (on-line transactional processing) and OLAP (on-line analytical processing). OLTP workloads are characterized by a mix of reads and writes to a few rows at a time, which is often latency-critical. On the contrary, OLAP applications are characterized by bulk sequential scans spanning a few columns of the database, such as computing aggregate values of specific columns. These two workloads are usually served by two different types of database systems, i.e. transactional processing and data warehouse systems. However, the explicit separation between OLTP and OLAP systems in IMDB suffers from several drawbacks.

One major shortcoming of this approach is that at least two copies of data are resident in memory. Despite several TBs of memory capacity in modern high-end server, RAM space is still a precious resource. Another serious limitation is the synchronization between two versions of data leads to the slow response time in analytics over real-time changing data [4]. IMDBs with substantial performance improvement have made it possible to process mixed OLTP and OLAP workloads (referred to as OLXP [10]) in a single database. In a nutshell, the integration of OLTP and OLAP database systems by means of in-memory technology has become a prevailing trend [4, 5, 6, 9, 11].

Unfortunately, the memory access efficiency is seriously degraded with a mixed workloads from IMDB supporting OLXP. The major reason is that OLTP and OLAP workloads generate two types of data access patterns. The former one is row-oriented access pattern that dominates in OLTP applications. The latter one is column-oriented access pattern, which dominates in OLAP applications. Conventionally, data in physical main memory is stored with a row-based layout that is friendly to row-oriented access pattern in OLTP. However, for OLAP with column-oriented access patterns, it results in intensive so-called *strided* memory accesses that degrade memory efficiency substantially due to poor DRAM row-buffer and cache utilization [12]. Note that such a problem is also present for OLTP in column-based data layout [13].

Without any doubt, performance of an IMDB is quite sensitive to the efficiency of accessing data in main memory [14]. 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 [12] to accelerate stride data access pattern by leveraging parallelism in DRAM chip level. However, this method has some limitations. First, it only exploits data already loaded in row buffers. It can not accelerate strided data access spanning DRAM rows. Second, GS-DRAM is not flexible enough for IMDB since only a specific set of access patterns (such as power-of-2 strided access) can be supported. It did not address the case of having different field width in the same tuple of IMDB. Third, its efficiency decreased with the size of a row in IMDB. If the tuple size of an IMDB is increased, there is actually less data in a column loaded in the row buffers so that the parallelism cannot be leveraged. Finally, the complexity increases with the number of multiple tables because multiple patterns may exist.

A theoretical solution is to design a memory architecture that enables both row and column accesses in a physical memory array like the transposable SRAM design [15, 16] proposed for multimedia and image-processing applications. Though it is possible to design such a DRAM module [17] (called RC-DRAM), it is impractical for the scenario of IMDB. The area overhead is even higher than database duplication due to asymmetric circuit layout of a DRAM device. Fortunately, we find that crossbar-based non-volatile memory (NVM) technologies [18], such as 3D XPoint, RRAM and PCM, are promising alternatives to support both row-oriented and column-oriented accesses with much lower area overhead. We design Row-Column-NVM (RC-NVM) that leverages the layout symmetry of crossbar-based NVM to enable flexible and efficient row and column accesses for IMDBs.

Contributions of this work are summarized as follows:

- We first propose a novel memory architecture called RC-NVM to support symmetric row and column accesses for IMDBs. We also present circuit-level analysis to prove that RC-NVM induces acceptable area overhead.
- We rethink the addressing schemes, data layouts, cache synonym, and coherence issues of RC-NVM in architectural level to make it applicable for IMDBs.
- We propose a group caching technique that combines the IMDB knowledge with the memory architecture to further optimize the system.

Experimental results show that the memory access performance can be improved up to 14.5X with only 15% area overhead, compared to conventional NVM. In fact, with our RC-NVM architecture, NVM not only provides several times of capacity increase than DRAM, but also achieves even better performance at the same time.

2. BACKGROUND

In this section, we first review the data layout 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 for IMDB applications. Finally, we use crossbar-based RRAM as an example to introduce the potential of RC-NVM design.

2.1 Data Layout Issue of IMDB

Relational databases represent data in a two-dimensional table of columns and rows. However, main memory is organized in single dimension, providing a flat memory addresses space that start at zero. The database storage layer must decide how to map the two-dimensional table structures to the linear memory address space.

The classical approach is a row-based layout ("row-store"). In this case, all fields of a tuple are stored consecutively and sequentially in memory. Two different access patterns for row-oriented and column-oriented operations are illustrated in Figure 1. The top half shows an OLTP query that selects a full table row, while the bottom half demonstrates an OLAP query that scans two columns. Note that the spatial locality cannot be utilized in column-oriented access, and the size of two fields is smaller than the cacheline size, which further reduce the memory bandwidth utilization.

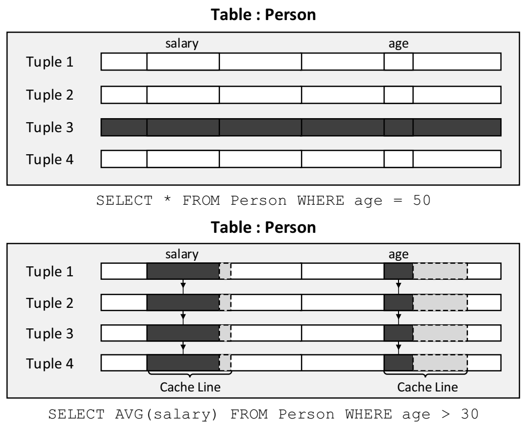


Figure 1: OLTP & OLAP accesses in row-store database.

Another layout, the columnar layout ("column-store"), is especially effective for set-based reads. In other words, it is useful for operations that work on many rows but only on a much smaller subset of all columns, as the values of one column can be read sequentially, e.g. when performing aggregate calculations. However, when performing operations on whole tuples, a row-based layout is beneficial. Currently, row-based layouts are widely used for OLTP workloads while column stores are widely utilized in OLAP scenarios like data warehousing.

However, a table only exists in either a row-based or a column-based layout, and both have their own weaknesses. OLAP on row-based layout, or OLTP on column-based layout will cause strided memory accesses and degrade the cache line and DRAM row buffer utilization, which is harmful for memory system performance. None of them can perfectly support both OLTP and OLAP (i.e. OLXP) workloads.

2.2 RC-DRAM Design

RC-DRAM, a.k.a. Dual-addressing DRAM [17], has been proposed to support both row-oriented and column-oriented memory accesses for DRAM. In the RC-DRAM array design, two transistors and one capacitor are employed to store one bit data, in contrast with one transistor and one capacitor per data bit in a conventional DRAM. Moreover, one extra word line and one extra bit line are used to support the column-oriented decoding.

In fact, a DRAM mat is the densest region in DRAM. Such dense mats are located in a bank repeatedly. The modification to DRAM mat in this design will lead to significant area-overhead, which is larger than 200% bit-per-area. Furthermore, some additional peripheral circuits, such as row decoder, column decoder, sense amplifier, etc., are required. Detailed area overhead results are shown in Figure 4. Such large area penalty is unacceptable since the capacity of RC-DRAM will be significantly reduced and not suitable for IMDB applications.

2.3 Crossbar-based NVM

In this section, we use RRAM as an example to explain why crossbar-based NVM is feasible for RC-memory design. The similar design can be extended to other crossbar-based technologies, such as PCM and 3D XPoint [19].

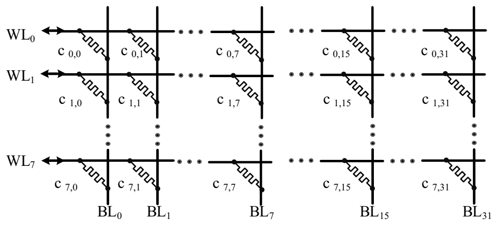


Figure 2: An 8×32 array of crossbar RRAM

An example of 8×32 crossbar array of RRAM is depicted in Figure 2. Each RRAM cell lies at the cross-point of word lines (WLs) and bit lines (BLs). Without introducing access transistors, these cells are directly interconnected with WLs and BLs via electrodes. Read and write operation can be performed by activating WLs and BLs with corresponding voltage. Note that RC-NVM is also feasible for an array with specific selector (e.g. FAST selector for RRAM [20] or OTS selector for PCM [21]) to control these cells.

To read out a row in the array, the target WL will be driven to read voltage V_{read}. The rest of WLs voltage are set to V_R, which is the read reference voltage. By keeping voltage of BLs being V_R with a current sensing amplifier, voltage across the unselected RRAM cells is equal to zero. Thus, the measured sensing current on each BL will be exactly the same as the one flowing through the access cell.

Since WLs and BLs are symmetric in such a crossbar array, reading out a column can be realized by simply exchanging behaviors of WLs and BLs. In Figure 2, we just need to exchange the voltages on WLs and BLs accordingly. Then, the current on each WL is sensed to read out the target cells on the target column.

For write operation, it requires two steps to write a row. First, the target WL and BLs will be tied to write voltage V_{write} (Gnd) and Gnd (V_{write}) for a SET (RESET) operation, respectively. Other WLs and BLs will be biased with half of V_{write}. Second, the target WL and BLs are applied with Gnd (V_{write}) and V_{write} (Gnd) for a RESET (SET) operation of remaining cells in the target row. Therefore, the write voltage V is fully applied across the full-selected cell(s). Other cells sharing the activated WL and BLs also bear partial voltage across them. *Similar to a read operation, we just need exchange the roles of BLs and WLs to write to a column.*

An important observation from the discussion above is that there is **no change** required to an NVM cell array to enable both row and column accesses because of the symmetry of the RRAM cell. Thus, compared to a RC-DRAM, we can achieve significantly smaller area overhead in a RC-NVM design.

3. RC-NVM CIRCUIT DESIGN

Figure 3 (a) shows the schematic view of a RC-NVM logic bank. Leaving RRAM array itself unchanged, extra peripheral circuitry is required to realize row and column accesses in the same bank. Both WL and BL are connected with dedicated decoder, sense amplifier (SA) and write driver (WD). The connection is controlled by multiplexers (MUXs) and control signals from the memory controller. In addition to the existing row buffer, a column buffer should be deployed for buffering column data.

As shown in Figure 3 (b), multiple mats in each bank is grouped into a logic structure called *subarray*. A subarray is the basic access unit of both row-oriented and column-oriented accesses. Figure 3 (c) provides a logical abstraction of a RC-NVM bank.

We take the row read operation as an example to explain how addressing is done in RC-NVM. Column read operation has the same flow with row and column. Given an access address, the global row decoder does the partial decoding to assert one global word line (GWL) and global block line (GBL). Then, local row decoder eventually generates 1-hot signal to assert the local WL (LWL). After that, column decoder selects the local bit line (LBL) and the cells on selected LBL are sensed out. Finally, SAs delivers the data to the row buffer through data line (DL). Such hierarchical decoding structure [22] can effectively reduce the decoding delay and power when the RC-NVM scales up.

If both row and column buffer *in the same bank* are in use, the data on the cross-point of the row and column are duplicated. This data duplication can incur coherence issue if the data is modified in one buffer while the other is not informed. To address this issue, we apply a restriction in this work: the row and column buffer **cannot** be active at the same time. If a row-column operation switch occurs, RC-NVM will close the active buffer and flush the data back, before it activates the new buffer. In this way, RC-NVM eliminates the data coherence problem with the overhead that bank must be reopened. However, our experimental results show that the design performs well and the overhead is marginal.

The comparison of area overhead between RC-DRAM and RC-NVM are given in Figure 4. X-axis shows the number of WLs and BLs in a single memory array, which represents

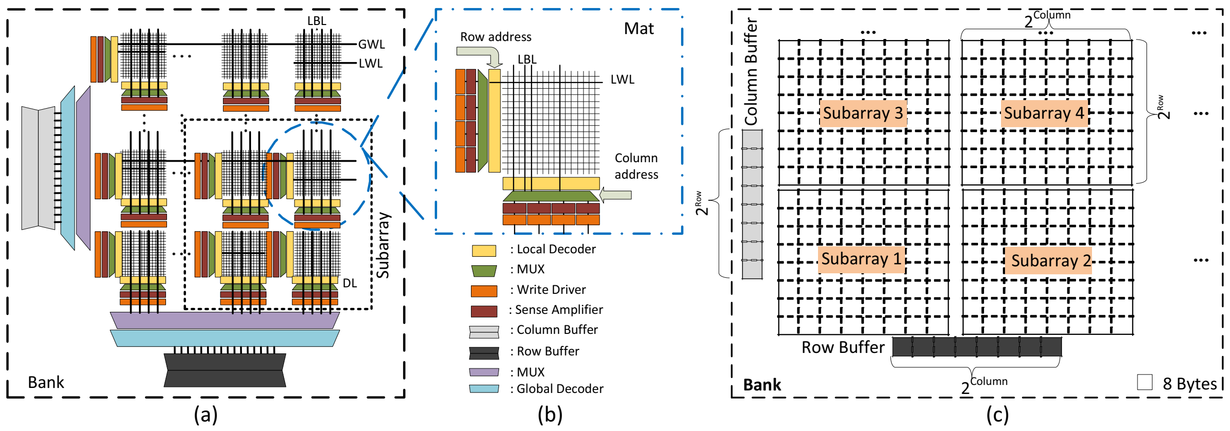


Figure 3: RC-NVM bank design.

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 NVM are from previous work [23]. We adopt a real DRAM module [24] and scale it to the same technology node for fair comparison. The total size of both memory chips are set to 4 GB.

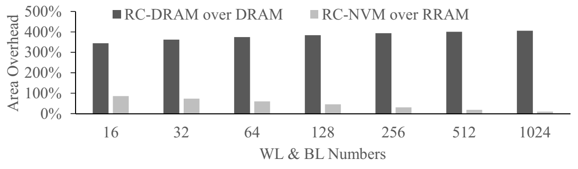


Figure 4: Area overhead of RC-DRAM and RC-NVM.

Figure 4 shows that the RC-DRAM always induces significant area overhead (more than 2X) over the original DRAM design. The area overhead is proportional to the number of WLs and BLs in an array. Therefore, RC-DRAM is not a good candidate for IMDB applications. On the contrary, the overhead of RC-NVM is significantly lower. Compared with RRAM, the proposed RC-NVM only requires extra peripheral circuitry while keeps the cell array intact. Thus, the overhead decreases as the area of cell array increases. As shown in Figure 4, the overhead drops to less than 20% when the numbers of WL and BLs are 512. Therefore, RC-NVM becomes more and more attractive with larger memory capacity.

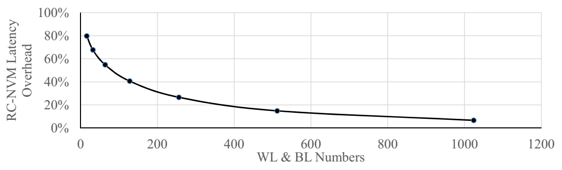


Figure 5: Latency overhead of RC-NVM.

The increase in area also induces longer latency mainly from wire routing overhead. Since more multiplexing transistors are added to the critical timing path, it also increases the read and write latency. The timing overhead of multiplexing, however, is trivial because the majority of latency comes

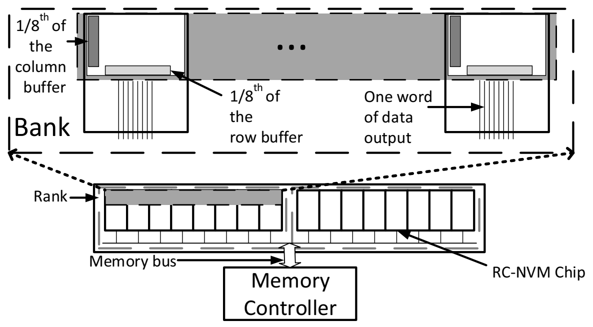


Figure 6: Overall architecture of an RC-NVM module.

from the cell access and wiring delay. To quantify the timing overhead, we run SPICE simulation and the results are shown in Figure 5. The timing overhead for RC-NVM is moderate. For example, when the numbers of WL and BLs are 512, the timing overhead is just about 15%.

4. ARCHITECTURAL DESIGN OF RC-NVM

In this section, we first present the overall architecture of RC-NVM memory systems. Then, we introduce the addressing scheme of RC-NVM to support both row- and column-oriented accesses. Next, we describe the cache architecture for RC-NVM. After that, the basic usage of RC-NVM and data layout are discussed from the perspective of IMDB.

4.1 Overall Architecture

The overall architecture is illustrated in Figure 6. The basic organization of RC-NVM main memory is similar to a traditional RRAM or DRAM design. It is also organized hierarchically as channel, rank, bank, etc.

In this example, there are two ranks on a DIMM device. 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- and column-oriented accesses. The granularity of row- and column-oriented accesses is 8 byte. 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 in each rank. Thus, the memory bus becomes 72-bit like common DRAM with ECC.

4.2 RC-NVM Addressing

In order to perform row- and column-oriented accesses in an IMDB, three requirements should be satisfied. First, we need to provide a dedicated addressing mode for each type of accesses. Second, an IMDB should explicitly control data layout on physical memory, which is similar to traditional databases using raw disks directly and enabling them to manage how data is stored and cached. Third, two new instructions are introduced to exploit the column-oriented accesses.

4.2.1 Dual Addressing Modes

As shown in Figure 7, we compare two different addresses for the same data in RC-NVM. Figure 7 (a) is a typical row-oriented address for a 32-bit conventional main memory. Figure 7 (b) 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 of the total 32-bit address.

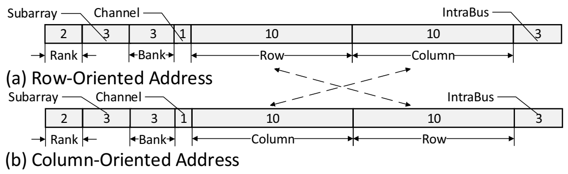


Figure 7: Address formats for (a) row-oriented and (b) column-oriented accesses.

Such addressing methods not only simplify design complexity of memory controller, but also make it easy to transfer addresses between two accesses modes. 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. Thus, it is simple to transfer a row-oriented address to a column-oriented address and vice versa.

4.2.2 Explicit Data Layout Control

In order to explicitly control physical data layout on RC-NVM, the IMDB can leverage the *huge-page* technique provided by mainstream operating systems [25], 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 [26].

As discussed in Section 4.1, the basic access unit is a subarray for both row-oriented and column-oriented accesses. Thus, given the address mapping design in Figure 7, an IMDB can explicitly control the data to be accessed in each row-/column-oriented access. Obviously, as long as the subarray bits, which include *row* bits and *column bits*, are allocated inside the lower 30-bit, an IMDB 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 discussion.

In a real case, 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. An IMDB can access these information with the help of OS so that the data layout is carefully organized to facilitate the row-/column-oriented accesses.

4.2.3 ISA Extension

In order to allow IMDB applications to utilize the column-oriented access in RC-NVM, we add two instructions, called *cload* and *cstore*. The details of these two instructions are shown as follows:

```
cload reg, addr
cstore reg, addr
```

where `reg` is the destination register, `addr` is the data address. Addresses in these two instructions are recognized by the memory controller, and sent to RC-NVM modules with an additional column-oriented signal. Similar to prior works, this can be in implemented by leveraging DDR interface. For example, DDR4 interfaces has two reserved address pins, thereby one of them can be used to send this signal [27]. Since traditional row-oriented addressing is still adopted in *load* and *store* instructions, other applications do not need to change.

4.3 Cache Architecture for RC-NVM

In this subsection, we focus on modification of CPU cache design to make it work with RC-NVM. First, we introduce how to store data with two different addresses in caches. Then, we discuss how to solve the data synonym problems in single-core and multi-core scenarios.

4.3.1 Caching Data with Dual Addresses

As introduced in Section 4.2.1, data in RC-NVM can be accessed with two different addresses, both of which can be used for cache addressing with conventional decoding circuitry. In order to differentiate two addresses in the cache, one extra status bit, called *orientation bit*, is added to each cache line. When data are loaded into cache with row-oriented addresses, the bit is set to ‘0’, otherwise it is set to ‘1’.

As shown in the example of Figure 8 (a), the 8-byte data can be accessed with addresses `0x0036a5b0` (row-oriented) and `0x0016cda8` (column-oriented). We have a cache with following configuration: 1) cache size is 64 KB, 2) cache block size is 64 byte, and 3) cache associativity is four. When the data are accessed with the row-oriented address, they are loaded into the cache together with other 56-byte data in the same NVM row. As shown in Figure 8 (b), data are placed in the corresponding cache entry and the orientation bit is set to ‘0’. Similarly, when accessed with the column-oriented address, they are loaded into cache together with other 56-byte data in the same column. Data are placed in the cache according to the column-oriented address with orientation bit set to ‘1’.

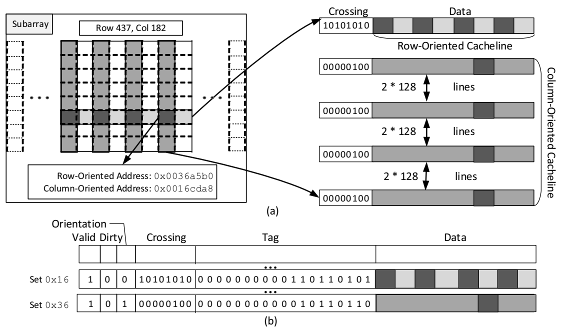

Subarray
Row 437, Col 182
Row-Oriented Address: 0x0036a5b0
Column-Oriented Address: 0x0016cda8
Crossing
Data
10101010
Row-Oriented Cacheline
00000100
2 * 128 lines
Column-Oriented Cacheline
(a)
Orientation
Valid
Dirty
Crossing
Tag
Data
Set 0x16 1 0 0 10101010 000000000001101101 01
Set 0x36 1 0 1 00000100 0000000000001011 0110
(b)

Figure 8: Illustration of cache architecture for RC-NVM.

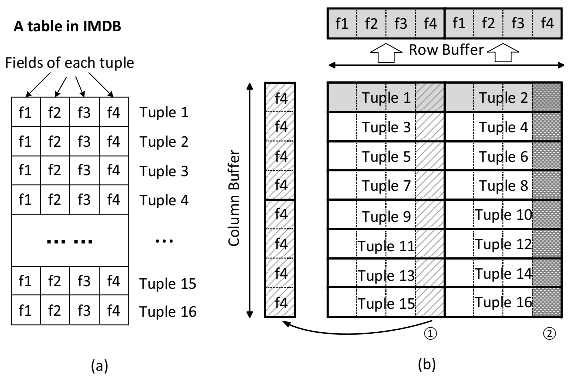


Figure 9: **An example of IMDB table and its layout in RC-NVM.**

4.3.2 *Cache Synonym in a Single-core Processor*

As shown in the previous example, same data may be loaded into the cache twice with both row-oriented and column-oriented addresses. It will result in the data synonym problem. A previous work has solved such a problem by separating the cache into two individual parts and employing a *WURF* cache coherence policy [17]. In this work, instead of partitioning the cache into two parts, we add one extra status bit for each 8-byte (i.e. granularity of data synonym) in the cache block. Thus, for a 64-byte cache block, we need 8 extra status bits, as shown in Figure 8 (b). These status bits are called *crossing bits* to represent data synonym caused by the row-column crossing blocks, shown in Figure 8 (a).

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, cache controller needs to check all potential cache lines that may cross with this one in the memory. For example, in Figure 8 (a), a 64-byte row-oriented cache block may be crossed with 8 column-oriented cache blocks in the RC-NVM. Thus, when the row-oriented cache block is loaded into the cache, 8 potential column-oriented cache blocks are checked. 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 are kept same. At the same time, the corresponding crossing bits are set to '1'. The status of the cache in this example is shown in Figure 8 (a).
- 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 modified 8-byte data is equal to '1', the corresponding duplicated data in the crossed cache block are updated at the same time.

4.3.3 *Solving Cache Synonym and Coherence Issues*

In a multi-core processor, the cache synonym problem also exists. At the same time, it will cause extra coherence problems. 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 make all updated blocks consistent in multiple cores or 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.

For both single-core and multi-core cases, 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 data replacement itself is a time-consuming process, the overhead is acceptable. Evaluation results of cache overhead are presented in Section 6.

4.4 Basic Usage of RC-NVM

In this section, we use a simplified scenario to demonstrate how to leverage both row-oriented and column-oriented accesses in RC-NVM. Figure 9 (a) illustrates the table in an IMDB 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 the memory, we use the term "tuple" to represent a row in the table of an IMDB. To simplify the discussion, the sizes of four fields are all set to 8 bytes in this example. We assume that this table is stored in a 512-byte RC-NVM subarray in a row-oriented way, as illustrated in Figure 9 (b). In this example, both the row buffer and the column buffer of this RC-NVM memory bank are set to 64 bytes.

Having this table in RC-NVM, we use two SQL examples to illustrate how row-oriented and column-oriented accesses work. The first one is a typical OLTP query for the row-oriented access, as shown in Figure 10. The SQL will fetch all tuples that satisfy the condition (`f3 < '1234'`). Obviously,

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

SELECT * FROM table WHERE f3 < '1234'

Figure 10: An OLTP SQL example: row-oriented access.

it is convenient to complete this SQL request with traditional row-oriented access. For instance, the first memory request loads both tuples, T1 and T2, into the row buffer. Then, the field f3 in each tuple is read out and compared. Then, data in T1 and T2 can be read out accordingly.

```
int sum = 0;
uint32_t col_addr_1, col_addr_2;
col_addr_1 = Row2ColAddr(&(table->tuple[0].f4));
col_addr_2 = Row2ColAddr(&(table->tuple[1].f4));
for (int i = 1; i <= 8; i++) {
  uint64_t f4_1 = column_load(col_addr_1);
  if (f4_1 < 4321)
    sum += f4_1;
  col_addr_1 += 8;
}
for (int i = 1; i <= 8; i++) {
  uint64_t f4_2 = column_load(col_addr_2);
  if (f4_2 < 4321)
    sum += f4_2;
  col_addr_2 += 8;
}
```

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

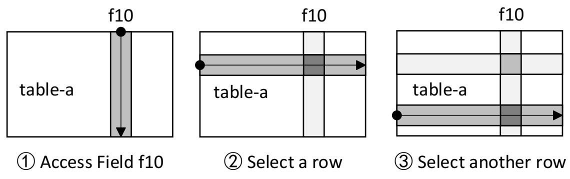
Figure 11: An OLAP SQL example: column-oriented access

The second example for column-oriented access is listed in Figure 11. This typical OLAP query fetches out all f4 fields in each tuple and adds them up. If we still use the traditional row-oriented access, all eight memory rows will be loaded into the row buffer sequentially to access fields in each tuple. However, since we have enabled column-oriented access in RC-NVM, this request is simplified significantly with only two column-oriented memory accesses to read out all fields required.

The third example that uses both row- and column-oriented accesses is illustrated in Figure 12. This query displays certain rows that meet the condition. Only a few rows that meet the restriction are selected. In RC-NVM, we can use column-oriented access to scan the f10 column to check whether the condition, f10 > x, is met. If a candidate is found, then the IMDB can issue a row-oriented access to retrieve the tuple. In this process, the data transmitted on memory bus are all effective, thus the utilization of memory bandwidth is improved.

4.5 Data Layout in RC-NVM

As we addressed in Section 4.2.2, we can explicitly control physical data layout in RC-NVM to facilitate data accesses. Thus, we can enable more flexible data placement in RC-NVM than in conventional DRAM. The goal is to store those



SELECT * FROM table-a WHERE f10 > x

Figure 12: An SQL example with both row- and column-oriented accesses

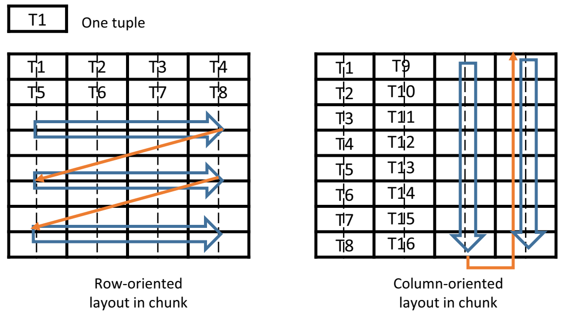

T1
One tuple
T1 T2 T3 T4
T5 T6 T7 T8
T9 T10 T11 T12 T13 T14 T15 T16
Row-oriented layout in chunk
Column-oriented layout in chunk

Figure 13: Two types of data layouts: (a) row-oriented layout and (b) column-oriented layout.

tables in IMDBs efficiently. In this subsection, we first introduce how to slice a table of IMDB into small chunks. Then, we discuss how to place these chunks in a RC-NVM.

4.5.1 *Slicing a Table into Chunks*

Since tables in IMDBs are normally very large, we need to slice them into multiple data chunks before placement. This is a common technique in database management to store large tables [28]. In this work, a chunk is defined as the rectangle unit of data placement in RC-NVM that can be fit into a subarray of RC-NVM. In other words, a table is sliced into chunks when its size is larger than a subarray of RC-NVM (i.e. 8 MB in this work) or the length of its tuple is longer than the row length in a subarray (i.e. 8 KB in this work). Note that the second case is really rare in real IMDB applications. After a table is sliced, we need to handle two data layout issues: intra-chunk data layout and inter-chunk data layout, which are introduced as follows.

4.5.2 *Intra-Chunk Data Layout*

We present two types of intra-chunk data layouts in a RC-NVM, which are called row-oriented layout and column-oriented layout in this work. They are friendly to row-oriented accesses and column-oriented accesses, respectively. These two types of data layouts are illustrated with two examples as follows.

A straightforward row-oriented data layout is illustrated in Figure 13 (a). The array represents a subarray of RC-NVM. Apparently, with such a data layout, the tuples in a chunk are continuous in a table. At the same time, their row-oriented addresses are also continuous according to the addressing method in Figure 7, which is similar to the data layout in conventional DRAM based main memory. Consequently, it is compatible to traditional IMDBs. In this layout, the row-oriented access 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 the column-oriented access will suffer from frequent column-buffer switching unless we do not care the accessing order within each column. To mitigate this problem, we further propose another column-oriented data layout, as shown in Figure 13 (b). Tuples are continuously placed in the vertical direction in a subarray. Thus, it is convenient to load the same field of multiple tuples with a single column-oriented access.

In fact, using a column-oriented data layout also outperforms the row-oriented counterpart with OLXP applications. The main reason is that a tuple is normally significantly shorter than the row size of a RC-NVM. Thus, in a row-oriented layout, the span of tuples in different memory rows is too large, which makes column-oriented accesses in OLAP quite inefficient. In addition, the access pattern of continuously scanning all tuples in an IMDB table is rare. Most SQL operations are composed of both row-oriented and column-oriented accesses. In addition, we will propose a dedicated data access optimization technique for column-oriented data layout in Section 5 to further improve its efficiency for OLXP.

4.5.3 Inter-Chunk Data Layout

At the beginning, all these subarrays are empty. Then, tables are created in run-time during IMDB operations. Since all IMDB tables have been sliced into multiple chunks, we need to figure out a 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 [29]. The goal of this algorithm is to minimize the number of subarrays that are used for at least one chunks. 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. GROUP CACHING OPTIMIZATION

We observe that the efficiency of column-oriented access is degraded when the data is required to be accessed with a specific order. As introduced in Section 4.1, the data width of a column-oriented access in RC-NVM is fixed (i.e., eight bytes in this work). However, the width of field in an IMDB table can vary. This fact may degrade efficiency of memory access, especially when a field width is larger than the column access width. Such a problem is called *wide field* access in this study.

An example is given in Figure 14. We assume that all data in field *Email* needs to be accessed. In this example, the *Email* field spanning two columns of RC-NVM is an indivisible whole. In column-oriented accesses shown in Figure 14 (b), only half of field data are read, which is meaningless for applications. Obviously, using column-oriented operations can traverse the whole field efficiently *only if* the access orders are not required. However, if the data access

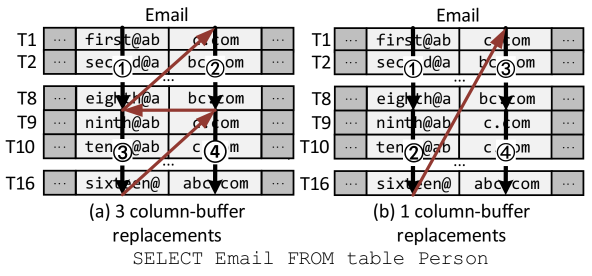


Figure 14: Wide field issue in column-oriented access

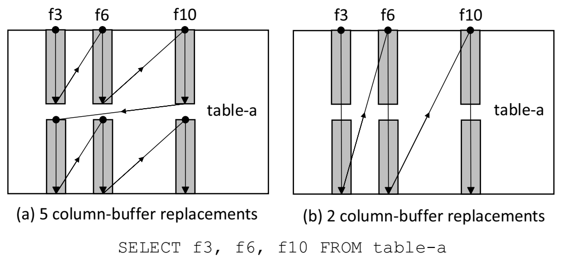

f3
f6
f10
table-a
(a) 5 column-buffer replacements
(b) 2 column-buffer replacements
SELECT f3, f6, f10 FROM table-a

Figure 15: Access multiple fields with column-oriented access

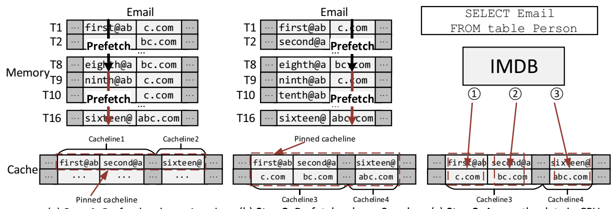
order from IMDB is strictly required as in the Figure 14 (a), column-oriented operation is inefficient. The reason is straightforward, each data access will generate a replacement of column-buffer. For instance, two extra column-buffer replacements occur in Figure 14 (a).

Another example can be found in Figure 15, the corresponding SQL is also listed. This query needs to read a few nonadjacent fields of a tuple in a specific order. In this example, the column-oriented access is also inefficient for the similar reason. Basically, whenever such a Z-style access order is required, the efficiency of column-oriented access is degraded. Unfortunately, the row-oriented access is also inefficient because only a small portion of data are hit in the row-buffer.

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 to CPU cache with column-oriented accesses. Then we can access the required data in any order with the help of the cache. We first modify the query optimizer in IMDB that converts SQL statements into memory access commands. When IMDB needs to access a wide field, or several fields that must be operated in row-order, it will generate corresponding group caching requests in advance. After the data is loaded into the CPU cache, IMDB can access the cached data using the column-major addresses.

One potential design issue is the unexpected cacheline eviction. It is possible that the cached group data of one thread are replaced by data accessed from the other threads before they are really accessed, especially in the multi-core environment. The solution for this cache thrashing problem is to use a well-known technique called cache-pinning [30]. The basic idea is to pin the data in the cache before they are accessed so that it is not evicted due to cache conflicts. We still use the wide field example mentioned above to demonstrate how this technique works.

As shown in Figure 16, when IMDB query planner needs to retrieve a wide field, it will generate a software column group caching command to read each segment of the wide field. Then the cachelines fetched will be pinned (Step 1 and 2). After the data are used by the IMDB in Step 3, these cachelines will be unpinned to release the space. With the help of group caching techniques, data in a rectangle region can be accessed in either row-oriented or column-oriented way. In terms of the shape of the region, query optimizer in IMDB can select accessing method (row or column) to minimize the number of memory accesses. If the width of the requested rectangle region is much less than the memory row length, which is a common case that fetches a few small-size fields in SQL SELECT statement, the column-oriented access will be the right choice.



(a) **Step 1**: Prefetch column 1 and pin the data (b) **Step 2**: Prefetch column 2 and pin the data (c) **Step 3**: Access the data in CPU cache

Figure 16: Illustration of group caching

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 optimized group caching size is not only related to the cache size but also depends on the data access pattern. We compare the performance of executing the SQL in Figure 14 and Figure 15 under different group caching sizes in Figure 23.

6. EVALUATION METHODOLOGY

In this section, we first introduce the experiment setup for simulation. Then, we provide details of workloads used for evaluation.

6.1 Experiment Setup

We use a cycle-accurate memory simulator, NVMain [31] integrated with gem5 [32] as our system simulator. We simulate an directory based MESI cache coherence protocol with Ruby in gem5. Based on the timing parameters of Panasonic's RRAM model [23], we modified NVMain to quantitatively evaluate the performance of the proposed RC-NVM. We also adopt Micron's DRAM [24] as another reference.

Our main system configuration is shown 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 our system is 4 GB. This configuration is matched to the address mapping scheme shown in Figure 7. The well-known FR-FCFS [33] is used as our scheduling policy.

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 scheduler [33]
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 byte, Capacity: 4 GB, Read access time: 29 ns, Write pulse width: 15 ns, 4 512*512 mats in a subarray

6.2 Workloads

We first describe the benchmarks used in our evaluation. As pointed out by prior work [34], there still lacks a standard OLXP benchmarks, we developed our own benchmark that is composed of queries that are common in enterprise workloads. [1] We first select a number of SQL queries to evaluate performance of RC-NVM. They are typical queries that perform small transactional operations (OLTP-style), as well as more complex, read-intensive aggregates on larger sets of data (OLAP-style). These queries composing our workloads are shown in Table 2. Queries Q1 to Q3 and Q8 to Q13 can be categorized as typical OLTP queries, while queries Q4 to Q7 can be categorized as OLAP-style queries. Queries Q14 and Q15 are used to evaluate the effect of group caching optimizations. 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.

7. EVALUATION RESULTS

In this section, we evaluate performance of RC-NVM and compare it with conventional RRAM and DRAM counterparts with different workloads.

7.1 Micro-benchmark Evaluation

Figure 17 shows the performance results of RC-NVM, RRAM, and DRAM with eight micro-benchmarks. The purpose of these micro-benchmarks is to scan an IMDB table with identical read/write operations. The table can be organized as row-oriented layout (labeled as L1) or column-oriented layout (labeled as L2), as shown in Figure 13. And there are two directions of scanning a table: row-oriented (labeled as row-read/write) and column-oriented (labeled as col-read/write). For conventional RRAM and DRAM designs, only row-oriented access is used for both directions of

[1]We are trying to make it open-source for future standard benchmarks. Code and workload description can be found anonymously at `https://github.com/RCNVMBenchmark/RCNVMTrace`.

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`

scanning. For RC-NVM, the row-oriented access and column-oriented access are used accordingly for different scanning directions.

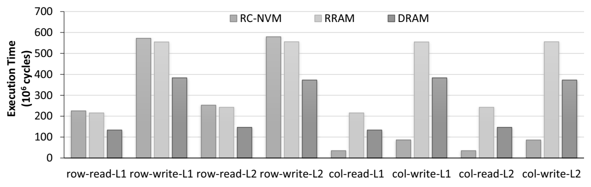


Figure 17: RC-NVM micro-benchmark results

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

7.2 Queries Evaluation

Figure 18 presents the experimental data 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% on average, respectively. All these results demonstrate 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 access, whose pattern is most 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. We also compare with results of using GS-DRAM [12]. Compared to GS-DRAM, the performance is improved by 2.37x on average. The reason of performance improvement with RC-NVM can be explained by a combination of factors as follows.

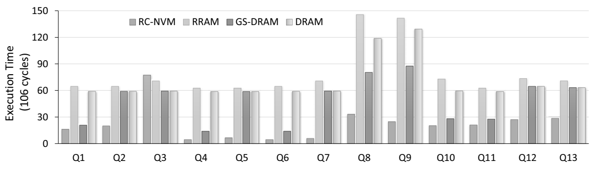


Figure 18: SQL benchmark results

First, by using both row-oriented access and column-oriented accesses, the total number of memory requests can be greatly reduced. As shown in Figure 19, the total number of memory accesses of RRAM and DRAM are the same since they can only use one-direction memory access. However, memory access numbers are greatly reduced in RC-NVM, even considering the effect of cache synonym and coherence. LLC misses are less than a third of those of DRAM on average. As a consequence, the total number of memory accesses to retrieve the effective data in RC-NVM decreases, compared with that in DRAM. In other words, IMDB using RC-NVM have two alternative ways to access the data, and it can select a best combination of access methods to fetch data at one time to improve the memory bus utilization. For GS-DRAM, memory accesses are only reduced when a specific set of access patterns (power-of-2 strided access) happens. These are cases for queries Q1, Q4, and Q6. For the cases like Q2, Q3, Q5, GS-DRAM cannot work. Thus, it shows no improvement over conventional DRAM.

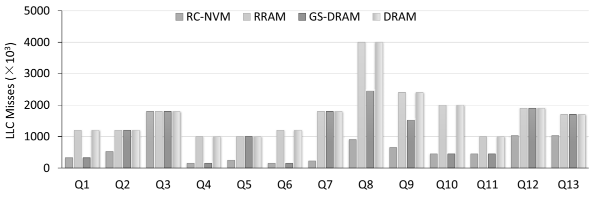


Figure 19: Number of memory accesses.

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. Figure 20 shows RC-NVM achieves a 38% decline in total buffer miss rate. Note that the misses of row-/column-buffer of RC-NVM are combined together as the total buffer miss rate. Note that the miss rate of column-buffer is not reduced after using GS-DRAM. It only scatters data into multiple rows and active them together.

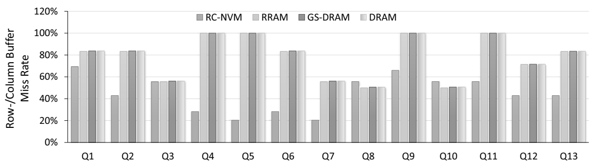


Figure 20: Comparison of row-/column-buffer miss rates.

Third, 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%. On average, the cache coherence overhead introduced by RC-NVM is 1.06%, 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 [12].

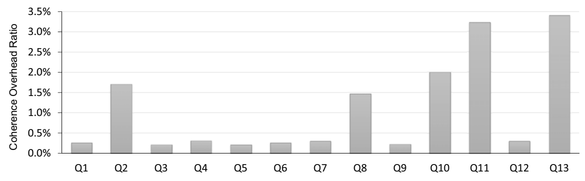


Figure 21: Cache synonym and coherence overhead.

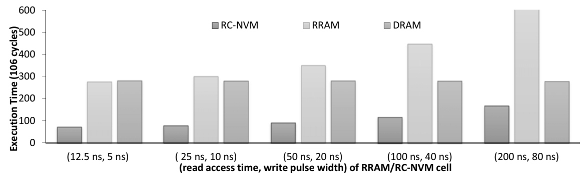


Figure 22: RC-NVM read latency sensitivity results

In previous experiments, we use the RRAM model from Panasonic's RRAM model [23]. In order to reflect the impact of different RRAM technologies on efficiency of RC-NVM, we perform a sensitivity analysis. As shown in Figure 22, we scale the read and write latency to different values and compare the average execution time results. We can find that using RC-NVM can still outperform DRAM even when the read and write latency are in the level of several hundreds of cycles.

7.3 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 Figure 23. The numbers on the legend indicate how many cache-lines are filled at one time. It is apparent from this figure that larger group caching size achieves better performance. For example, we can achieve a 15% performance improvement when the group caching length is set to 128 cachelines for each column. The estimated cache space we need for Q14 and Q15 are 32K and 24K, respectively.

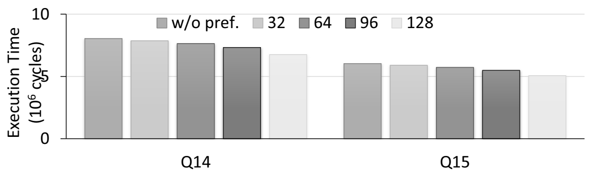


Figure 23: Impact of Group Caching optimization

8. RELATED WORK

We believe that this is the first work that shows the use of a row-column-accessible RAM in in-memory database. In this section, we discuss the prior works that aim to improve the efficiency of OLTP and OLAP queries in IMDB, and enhance the performance of the memory system.

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 [38,39,40]. 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 [41, 42], and can achieve better data compression [43], but has a negative impact for OLTP queries that need to operate on the row level [5, 9, 38].

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, PAX [44] stores data from multiple columns only within a page, and uses a column-wise data representation for those columns. SAP HANA [9, 45] 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 [34]. 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.

High Performance Memory Architectures. Many previous work introduce new DRAM architectures for either achieving lower DRAM latency or higher parallelism. Redundant Memory Mapping (RMM) [46] is a virtual address translation mechanism that improves performance of accessing large memories. SALP [47] exploits the subarray-level parallelism to mitigate the performance impact of bank conflicts in DRAM. Our mechanisms are orthogonal to these works, and can be applied together with them to further increase memory system performance.

Recently, there has been extensive research work about using these NVMs as alternatives of DRAM or together with DRAM as a hybrid architecture [48, 49, 50]. For example, Intel has announced 3D XPoint based product that can be used as RAM [51]. And one of its target killer applications is IMDB because it can provide almost 10X capacity over DRAM main memory. Prior work mainly focuses on NVM advantages, such as non-volatility, high storage density, and low standby power. Yet the layout symmetry of crossbar-based NVM is not exploited.

Dual-addressing memory [17] (RC-DRAM) provides row-major and column-major memory access patterns. GS-DRAM improves the performance of the strided access by changing the DIMM organization [12]. Compared to them, RC-NVM facilitate IMDB to efficiently perform both row- and column-oriented accesses with considerable flexibility and small overhead.

9. CONCLUSION

In order to improve memory access efficiency of OLXP workloads in IMDBs, we propose a novel architecture called RC-NVM. The basic idea is to leverage the symmetric array structure of cross-bar based NVM technologies, such as PCM and RRAM, to enable both row-oriented and column-oriented memory accesses with moderate density loss. With a minor extension to the ISA and the help of huge-page technique, IMDBs can explicitly issue proper direction of memory accesses according to the data layout and database request patterns. Thus, the memory performance for OLXP workloads is significantly improved because both the number of memory requests and the memory buffer miss rates can be reduced substantially. We also propose a group caching technique to solve the problem of ordered data accesses and further improve performance at the same time. After using RC-NVM architecture with IMDBs, 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 for IMDBs.

10. ACKNOWLEDGMENTS

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

11. REFERENCES

[1] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner, "MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, (New York, NY, USA), pp. 479–490, ACM, 2006.

[2] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-store: A High-performance, Distributed Main Memory Transaction Processing System," *Proc. VLDB Endow.*, vol. 1, pp. 1496–1499, Aug. 2008.

[3] M. Stonebraker and A. Weisberg, "The VoltDB Main Memory DBMS.," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[4] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP amp;OLAP main memory database system based on virtual memory snapshots," in *2011 IEEE 27th International Conference on Data Engineering*, pp. 195–206, Apr. 2011.

[5] M. Grund, J. KrÃijger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "HYRISE: A Main Memory Hybrid Storage Engine," *Proc. VLDB Endow.*, vol. 4, no. 2, pp. 105–116, 2010.

[6] T. Lahiri, M.-A. Neimat, and S. Folkman, "Oracle TimesTen: An In-Memory Database for Enterprise Applications.," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 6–13, 2013.

[7] J. LindstrÃűm, V. Raatikka, J. Ruuth, P. Soini, and K. Vakkila, "IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability.," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 14–20, 2013.

[8] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling, "Hekaton: SQL Server's Memory-optimized OLTP Engine," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, (New York, NY, USA), pp. 1243–1254, ACM, 2013.

[9] V. Sikka, F. FÃd'rber, W. Lehner, S. K. Cha, T. Peh, and C. BornhÃűvd, "Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, (New York, NY, USA), pp. 731–742, ACM, 2012.

[10] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. FÃd'rber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner, "Towards Scalable Real-time Analytics: An Architecture for Scale-out of OLxP Workloads," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1716–1727, 2015.

[11] T. MÃijhlbauer, W. RÃűdiger, A. Reiser, A. Kemper, and T. Neumann, "ScyPer: Elastic OLAP throughput on transactional data," in *Proceedings of the Second Workshop on Data Analytics in the Cloud*, pp. 11–15, ACM, 2013.

[12] 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 *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, (New York, NY, USA), pp. 267–280, ACM, 2015.

[13] H. Plattner and A. Zeier, *In-Memory Data Management: Technology and Applications*. Springer Science & Business Media, May 2012. Google-Books-ID: HySCgzCApsEC.

[14] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the Memory Wall in MonetDB," *Commun. ACM*, vol. 51, pp. 77–85, Dec. 2008.

[15] J. s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, Sept 2011.

[16] B. A. Chappell, Y.-C. Lien, and J. Y. Tang, "Transporsable memory architecture," July 1989. US Patent App. US4845669 A.

[17] Y. H. Chen and Y. Y. Liu, "Dual-addressing memory architecture for two-dimensional memory access patterns," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pp. 71–76, Mar. 2013.

[18] D. B. Strukov and R. S. Williams, "Four-dimensional address topology for circuits with stacked multilayer crossbar arrays," *Proceedings of the National Academy of Sciences*, vol. 106, pp. 20155–20158, Jan. 2009.

[19] "3D XPoint Technology." `https://www.micron.com/about/emerging-technologies/3d-xpoint-technology`.

[20] 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 *2014 IEEE International Electron Devices Meeting*, pp. 6.7.1–6.7.4, 2014.

[21] S. R. Ovshinsky, "Reversible electrical switching phenomena in disordered structures," *Phys. Rev. Lett.*, vol. 21, pp. 1450–1453, Nov 1968.

[22] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. USA: Addison-Wesley Publishing Company, 4th ed., 2010.

[23] 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 8mb multi-layered cross-point ReRAM macro with 443mb/s write throughput," in *2012 IEEE International Solid-State Circuits Conference*, pp. 432–434, 2012.

[24] "DDR3 SDRAM." `https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/4gb_ddr3_sdram.pdf`.

[25] "Hugepages." `https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt`.

[26] "Configuring hugepages for oracle database." `https://docs.oracle.com/cd/E37670_01/E37355/html/ol_config_hugepages.html`.

[27] "DDR4 SDRAM STANDARD." `http://www.jedec.org/standards-documents/docs/jesd79-4a`.

[28] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An Overview of the HDF5 Technology Suite and Its Applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, (New York, NY, USA), pp. 36–47, ACM, 2011.

[29] S. Fujita and T. Hada, "Two-dimensional on-line bin packing problem with rotatable items," *Theoretical Computer Science*, vol. 289, no. 2,

pp. 939–952, 2002.

[30] 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," Aug. 13 2015. US Patent App. 13/976,181.

[31] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems," *IEEE Computer Architecture Letters*, vol. 14, pp. 140–143, July 2015.

[32] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, and others, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[33] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *SIGARCH Comput. Archit. News*, vol. 28, pp. 128–138, May 2000.

[34] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the Archipelago Between Row-Stores and Column-Stores for Hybrid Workloads," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), pp. 583–598, ACM, 2016.

[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 *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, (Washington, DC, USA), pp. 39–50, IEEE Computer Society, 1998.

[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 *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, (Washington, DC, USA), pp. 15–26, IEEE Computer Society, 1998.

[37] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a Modern Processor: Where Does Time Go?," in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, (San Francisco, CA, USA), pp. 266–277, Morgan Kaufmann Publishers Inc., 1999.

[38] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. Row-stores: How Different Are They Really?," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 967–980, ACM, 2008.

[39] S. Manegold, P. A. Boncz, and M. L. Kersten, "Optimizing Database Architecture for the New Bottleneck: Memory Access," *The VLDB Journal*, vol. 9, no. 3, pp. 231–246, 2000.

[40] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented Database Systems," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1664–1665, 2009.

[41] H. Plattner, "A Common Database Approach for OLTP and OLAP Using an In-memory Column Database," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, (New York, NY, USA), pp. 1–2, ACM, 2009.

[42] M. Kaufmann and D. Kossmann, "Storing and Processing Temporal Data in a Main Memory Column Store," *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1444–1449, 2013.

[43] C. Lemke, K.-U. Sattler, F. Faerber, and A. Zeier, "Speeding Up Queries in Column Stores: A Case for Compression," in *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK'10, (Berlin, Heidelberg), pp. 117–129, Springer-Verlag, 2010.

[44] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving Relations for Cache Performance," in *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, (San Francisco, CA, USA), pp. 169–180, Morgan Kaufmann Publishers Inc., 2001.

[45] V. Sikka, F. FÃď'rber, A. Goel, and W. Lehner, "SAP HANA: The Evolution from a Modern Main-memory Data Platform to an Enterprise Application Platform," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1184–1185, 2013.

[46] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. ÃIJnsal, "Redundant Memory Mappings for Fast Access to Large Memories," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 66–78, ACM, 2015.

[47] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, (Washington, DC, USA), pp. 368–379, IEEE Computer Society, 2012.

[48] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for nvm+dram hybrid main memory," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, (Berkeley, CA, USA), pp. 14–14, USENIX Association, 2009.

[49] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-MontaÃśo, "Improving read performance of phase change memories via write cancellation and write pausing," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–11, Jan 2010.

[50] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3d stacked mram l2 cache for cmps," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 239–249, Feb 2009.

[51] "Intel Optane Memory FAQ." `http://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory-faq.html`.