Energy-Efficient SQL Query Exploiting RRAM-based Process-in-Memory Structure

Yuliang Sun, Yu Wang, Huazhong Yang
Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList),
Tsinghua University, Beijing, China
e-mail: yu-wang@mail.tsinghua.edu.cn

***Abstract*—With the coming of 'Big Data' era, high-energy-efficiency database is demanded for the Internet of things (IoT) application scenarios. The emerging Resistive Random Access Memory (RRAM) has been considered as an energy-efficient replacement of DRAM for next-generation main memory. In this paper, we propose an RRAM-based SQL query unit with process-in-memory characteristic. A storage structure for database in RRAM crossbar array is proposed, which avoids redundant data transfer to cache and reduces cache miss rate compared with the storage method in DRAM for in-memory database. The proposed RRAM-based SQL query unit can support a representative subset of SQL queries in memory, and thus further reduce the data transfer cost. Simulation results show that the energy efficiency of the proposed RRAM-based SQL query unit is increased by 4 to 6 orders of magnitude compared with the traditional architecture.**

I. Introduction

With the development of green computing research, the database community focuses not only on the performance, but also the energy efficiency [1]. Especially for the database used in the Internet of Things (IoT) scenarios like wireless sensor network, the energy consumption becomes the main design considerations due to the limited supply of energy [2].

The data transfer costs the main part of energy consumption, which meets a "memory wall" bottleneck in von Neumann architecture [3]. An intuitive way to alleviate energy consumption is to reduce the volume of data being transferred from memory and/or storage to CPUs. Row-oriented (tuple-oriented) and column-oriented (attribute-oriented) storage structures are used in database applications to make the data match the cache line scheme and reduce cache miss. However, since a query in database applications may require the data from a row (tuple) and a column (attribute) simultaneously, the cache miss rate increases if the query does not match the storage structure [4].

The resistive random access memory (RRAM) is an emerging Non-Volatile Memory (NVM) device with high storage density and access speed [5]. The bi-polar characteristic of RRAM cell and the crossbar structure of RRAM array provide the potential to read data using both the row-oriented and the column-oriented methods. Based on this characteristic, using RRAM crossbar in database can potentially overcome the problem of the cache miss in traditional storages.

Near-memory processing is an alternative to reduce the data transfer in database applications on traditional hardware platform. Some computing circuits are designed in memory devices, and therefore this structure can transfer only the computation results instead of the raw data [6]. This method reduce the volume of data being transferred from memory to CPUs, but all the raw data still need to be read from memory to computing circuits. Researchers have shown that crossbar array of RRAM supports the computing of matrix operation, which means the computation can be processed in memory and the energy efficiency can be further improved [7]. Various RRAM-based computation systems have been proposed and fabricated [7]–[11].

This work was supported by National Natural Science Foundation of China (No.61373026, 61622403), Huawei Technologies Co. Ltd, and Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608).

SQL queries, which are the most popular operations in database applications, also contain computations that can be regarded as matrix operations. This observation motivates us to process the SQL query by the processing-in-memory (PIM) structure of RRAM crossbar and boost the energy efficiency of database.

In this paper, we propose an RRAM-based SQL query unit for database applications utilizing the PIM structure. The main contributions of this paper contain:

1) We propose a structured storage scheme for RRAM-based database. This storage structure can support straightforwardly reading rows and columns from a table in database applications, which avoids unnecessary cache miss compared with traditional architecture.
2) We propose an RRAM-based multi-mode SQL query unit, which implements restriction, projection, and aggregation operations with PIM characteristic. The proposed unit also supports the creation, update, and deletion operations that can maintain the structured storage scheme.
3) For large database table, a correlation-specific splitting method is proposed to store the table in multiple RRAM crossbars and minimize the unit-to-unit communication.
4) Simulation results show that the proposed SQL query unit obtains 4 to 6 orders of magnitude in energy efficiency compared with in-memory database solution on traditional architecture.

The rest of this paper is organized as follows: Sec. II provides the preliminaries and Sec. III introduces the motivation of this work. The proposed structured storage scheme in RRAM that supports a representative subset of SQL query operations is introduced in Sec. IV. Sec. V provides the structure of RRAM-based SQL query unit. Sec. VI presents the simulation results and Sec. VII concludes this work.

II. Preliminaries

A. Operations in SQL Query

In relational databases, a data record is defined as a $tuple$ with one or more $attribute$s. Many tuples are organized as a $table$, in which each row represents a tuple while each column represents an attribute. Structured query language (SQL) is used to manage the data stored in such tables with CRUD (Create, Read, Update, and Delete). When it comes to SQL, we choose a subset of typical operations that are widely used in query processing and propose an RRAM-based query unit design. The SQL query operations we support in this work are restriction, projection and aggregation.

Restriction: Restriction means that a query picks up certain tuples, or certain rows in a table only if the tuples meet the given requirements. The requirements can be algebraic logic or boolean logic constraint condition. Restrictions are expressed as WHERE-statement in the SQL syntax.

Projection: Projection is to pick up a set of specific attributes in a tuple, or specific columns in a table. Projections are expressed as SELECT-statement in the SQL syntax.

Aggregation: Aggregation is to summarize certain properties of several attributes in a group of tuples. For example, the aggregation function SUM() in SQL is to calculate the sum of values.

B. Related Work

1) Near-Memory Processing for SQL Query: Researchers have proposed several hardware accelerating solutions for SQL query tasks using application-specific platforms. The acceleration structure of some major SQL operations is proposed on parallelism computing platforms like GPUs [12] and FPGAs [13]–[16]. Wu et al. [17] designed and evaluated a performance-and-energy-efficient database processing unit called Q100. Q100 contains a collection of fixed-function ASIC tiles, each of which implements a relational operator. Jo et al. focuses on the architectural design and proposed YourSQL [6], a database system architecture that leverages in-storage computing (ISC) for query offloading to commodity NVMe SSDs. These solutions have shown great energy efficiency and speed improvement of near-memory processing on database and SQL query applications. However, limited by the traditional CMOS technology and von Neumann architecture, the large amount of data transferring between memory and associated accelerating platform still leads to high energy consumptions.

2) NVM in Database: To reduce the consumptions of data storage, Non-volatile Memory (NVM) technologies have been utilized in database applications. The method to efficiently integrated Non-Volatile Memory into the data management stack have been well discussed [18] to support transactions and to ensure persistence and recovery of data. The NVM programming model and the corresponding library for NVM database system recovery optimization has been proposed [19]. Based on these results, kinds of NVM can be integrated into database engines and the performance can be optimized by the cooperation of DRAM and NVRAM [20]. These results have demonstrated the availability of NVM techniques in database applications, which motivates us to explore the potential of further boosting the energy efficiency of SQL query applications by utilizing the bi-polar and PIM characteristic of RRAM crossbar.

III. Motivation

To illustrate the motivation of using RRAM-based structure in database applications in this work, explanations in two aspects are given in this section. 1) The traditional row- and column-oriented database storage structures in memory have inevitable redundant data loading from memory to cache and thus suffer from cache miss and low energy efficiency. The bi-polar characteristic of RRAM and crossbar structure provide a potential solution of a new storage schema to overcome this problem. 2) RRAMs have the capability of matrix computation, in which there is a potential for in-memory SQL query processing.

A. Storage Methods of In-Memory Database

Supposing that Fig. 1 is a row-oriented database table in DRAM, data is stored by one row after another: $tuple_1 = (A_1, B_1, C_1, ..., H_1)$, and then $tuple_2, tuple_3$.... Based on the knowledge of the existing memory hierarchy, when data is loaded, for example when A_1 is loaded, the following data B_1, C_1, that are stored just behind A_1 will also be loaded into a cache line. For convenience, the width of cache line is the same as the width of a tuple in Fig. 1.

For example, when a query "select $*$ from $Table$ where $A+B+C+D+E+F<100$" is executed, A_1, B_1, C_1, D_1, E_1, and F_1 in $tuple_1$ are asked and the whole $tuple_1$ will be asked if the condition "$A+B+C+D+E+F<100$" can be met. With the cache mechanism, when A_1 is loaded and accessed, $B_1, C_1, D_1, E_1 and F_1$

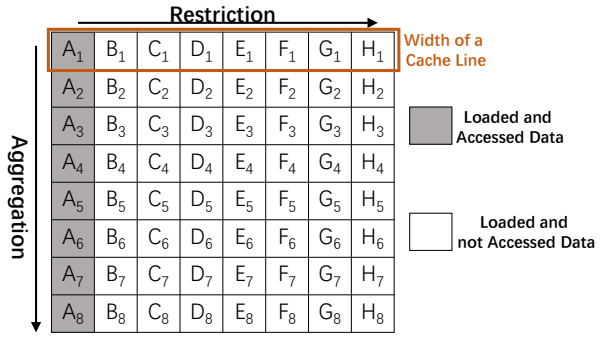

Fig. 1: An example to illustrate the cache miss problem with query "select A from $Table$ where $A<10$".

are loaded in the cache line and then accessed together. There is no additional cache miss in this example.

However, cache misses become inevitable with the query "select A from $Table$ where $A<10$". The query asks for attribute A in $Table$. When A_1 is loaded and accessed, A_2 is not in the same cache line and has to be loaded into cache again, that is how cache miss happens.

In the row-oriented databases, if a query asks for a row in a table, when the first data in a row is loaded, the following data are also loaded in the cache line. If a column is requested, cache miss happens since the data in one column dispersedly locate in the memory. The situation is just the opposite when it comes to the column-oriented databases. Cache misses happen when queries ask for a row in column-oriented databases.

Cache miss leads to extra loaded requests to DRAM and redundant data loaded in cache which is not accessed by queries. To achieve high energy efficiency in the system, cache misses should be avoided as much as possible.

However, cache miss is somehow inevitable for a complex query processing in a pure row or column store, because there must be both requests for rows and columns of tables in the query processing [4]. For example, an aggregation operation accesses a specific attribute, while a restriction operation accesses several attributes of a tuple, as shown in Fig. 1. There is also a hybrid data storage model in [4] using the vertical partition technology, but the topology of the data storage heavily depends on the workload. Reorganization can be a very expensive task once the workload is changed.

Therefore, it would be much more efficient if a new storage schema can be found in which we can get both columns and rows straightforwardly from a table in database applications. It inspires us that with the bi-polar characteristic and crossbar structure, an RRAM crossbar array can implement reading rows as well as columns in a table. The analysis and design of the RRAM-based structured storage are introduced in detail in Sec. IV-A.

B. RRAM and RRAM-based Computing for Database

RRAM is a passitive two-terminal emerging non-volatile memory device that stores data by the variable resistance of device. Fig. 2(a) shows the crossbar structure of RRAM array with word-lines and bit-lines, where every word-line is connected with every bit-line via an RRAM cell. Some RRAM cells support two current directions as a two-terminal device, which are referred as the bi-polar RRAM cells [5]. This observation motivates us to use the bi-polar RRAM cells in the crossbar, and we can also read out data from two directions. Specifically, we can read a row of data from bit-line, or regard the word-line as output port to read a column. This structure potentially solve the limitation caused by the fixed output port (the cache line) in traditional memory architecture.

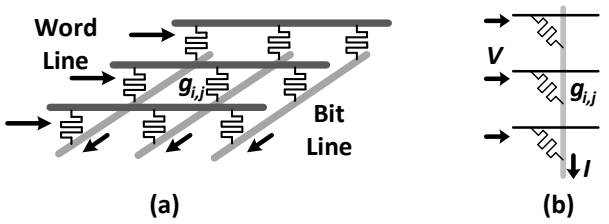

Fig. 2: RRAM crossbar and RRAM-based computing.

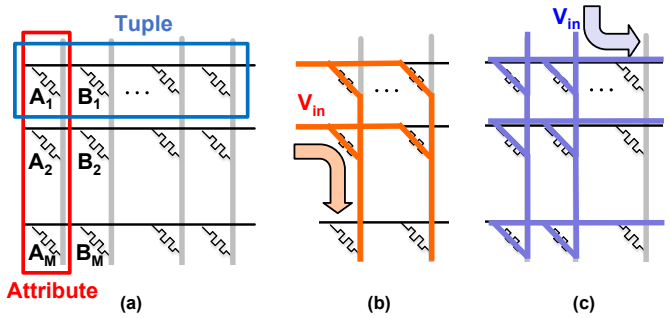

Fig. 3: Processing-in-memory structure for SQL query constraints. (a) Structured table storing scheme. (b) Column-wise PIM direction. (c) Row-wise PIM direction.

Moreover, when we apply voltages onto all the word-lines as inputs, the crossbar structure can perform matrix-vector multiplication in analog domain. According to Ohm's law, the conductance of RRAM equals to the product of input voltage and cell conductance. If $V_{(i)}$ is the input voltage of the ith word-line, $g_{(i,j)}$ is the conductance of RRAM cell connecting the ith word-line and the jth bit-line, and the current passes this RRAM cell is $I_{(i,j)}$, we have $I_{(i,j)} = V_{(i)} * g_{(i,j)}$. As Fig. 2(b) shows, the currents on the same bit-line are merged together, just like the accumulation step in vector-vector multiplication. Therefore, the entire RRAM crossbar accomplishes the function of $\boldsymbol{I} = \boldsymbol{V} \cdot \boldsymbol{g}$, where $\boldsymbol{I}$ is the vector of output current, $\boldsymbol{V}$ is the vector of input voltages, and $\boldsymbol{g}$ is the conductance matrix.

Researchers have proposed kinds of RRAM-based computing structures for different applications, including dot-product engine [8], image convolution [9], and brain-inspired neural computing algorithms [7], [10], [11]. These results have validated the energy efficiency of RRAM-based PIM structure, which provide great potential to explore this structure in database applications with large data transfer amount.

In this work, the PIM characteristic of RRAM is leveraged on database applications. As introduced in Sec. II-A, a large subset of SQL queries is linear operation. RRAM seems to be a suitable alternative to execute these queries for its advantages in processing matrix. The design to support different operations in database on RRAM is introduced in Sec. IV-B, Sec. IV-C, and Sec. IV-D.

IV. RRAM-based PIM Scheme for SQL Query

In this section, we introduce the structured storage scheme of an RRAM crossbar in our work and the method to map the SQL operations to a crossbar.

A. Structured Storage

As shown in Fig. 2(b) and Fig. 3(b), when we input the computing signals to multiple RRAM rows, the dot-product computing function is processed on the RRAM cells in the same column, which is referred as the column-wise computing. If we change the input ports to the RRAM columns, as shown in Fig. 3(c), the computing is processed on the RRAM cells in the same row, which is called the row-wise computing. In this way, the input voltages are applied onto all the bit-lines and the currents on the same word-line are merged together. Limited by the wire connections in the RRAM crossbar, only the column-wise computing and the row-wise computing are supported by the RRAM-based PIM scheme. Therefore, if we locate the data onto RRAM cells like a traditional row- or column-oriented storage, in which the data are not tightly aligned, the PIM structure is difficult to be utilized.

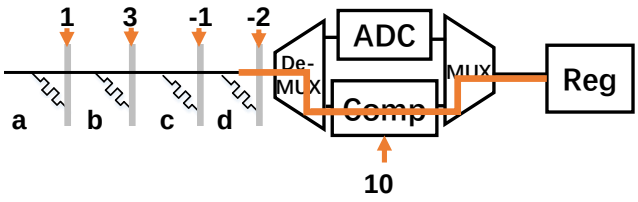

Fig. 4: An example for PIM restriction operation.

We propose a structured storage scheme for RRAM-based SQL query with PIM characteristics. The computation requirements in SQL query contain two major classes: 1) verifying a constraint of query using multiple attribute data in the same tuple, and 2) verifying a constraint of query using the same attribute in multiple tuples. Therefore, the RRAM crossbar need to support two kinds of dot-product computation. First, the dot-product computation at the RRAM cells storing the same attribute of different tuples. Second, dot-product computation at the RRAM cells storing different attributes of the same tuple.

Motivated by this observation, we map the data to RRAM cells in the same location as the data in the database tables. As shown in Fig. 3(a), each RRAM row stores a tuple, and each RRAM column stores the same attribute of multiple tuples. In other words, an RRAM row stores a row of a table, and an RRAM column stores a column of a table. In this way, we can perform dot-product computation of an attribute by the column-wise direction, as shown in Fig. 3(b); and perform dot-product computation of a tuple by the row-wise direction, as shown in Fig. 3(c). For example, for a table storing transcripts, we store the scores of multiple courses of a student in an RRAM row, and store the scores of all students in multiple RRAM rows in the same RRAM crossbar. Therefore, the average score of a course can be calculated by the column-wise direction, and the grade point average of a student by the row-wise direction. The detailed PIM method for each kind of query operation will be illustrated in the following parts of this section.

B. Restriction

The SQL standard supports a wide variety of restrictions working on different data types. In this work, a representative subset of the restrictions is chosen and supported on the RRAM cells as following:

- $+,-,\times$;
- $>,<,=,\geq,\leq,\neq$;
- AND,OR,NOT,XOR,NAND,NOR

The operators supported by the RRAM cells in this work can be implemented by the dot-product computations, the comparators, and the logical gate design in the peripheral circuit.

The addition operators, the subtraction operators, and the multiplication operators with constants can be seen as the linear combination of the contents in the RRAM cells. In this way, we can implement these operations with the dot-product computation by the row-wise direction. Since the RRAM-based PIM structure cannot directly support the multiplication between two attributes, we transfer the two

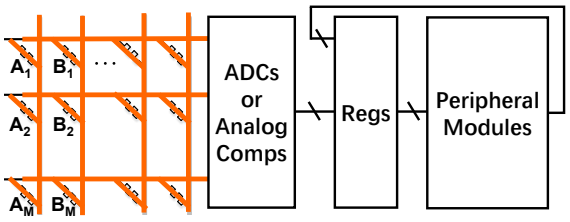

Fig. 5: Restriction operation implemented on RRAM crossbar.

attributes into CPU to perform the multiplication and the following operations.

Taken the query "select $*$ from $Table$ where $(a+3*b)-(c+2*d) > 10$" as an example, the computation of the restriction of a crossbar is shown in Fig. 4. In addition, if an attribute is not used in the restriction, we can perform the calculation by inputing a zero signal to fix the voltage at ground, or closing the input port and make the voltage be floating. The float port in the crossbar structure causes a sneak-path problem [21], which means the non-selected attributes will influence the calculation. Therefore, we use zero input signal to avoid sneak-path, and the zero voltage is provided by a local to reduce the energy consumption on DAC, as shown in Fig. 4.

As shown in Fig. 5, the weights of a restriction are input from the column ports, and the calculation results of all the RRAM cells in the same row are merged taking advantage of the PIM structure. If "$>$", "$\geq$", "$<$", or "$\leq$" is used in the restriction, we use an analog comparator at the row-output port to obtain the comparison result and reduce the hardware overhead by eliminating ADCs. If the output result needs to be processed by "$=$" or boolean functions, we use an ADC at a row-output port to transform the intermediate results into digital signals, which can be buffered in the registers. After that, the other operations in the restriction can be performed in a reconfigurable peripheral module using digital circuits.

C. Projection

Projections are used to select a specific group of columns (attributes) in a table. Based on the structured storage method, the dot-production computation by the row-wise direction is used, when selection operations ask for specific columns. The input voltage vector is set to 1 in the target column's position and 0 in the other positions. In this way, the RRAM cells can output 1 column in 1 dot-product computation. For example, if the first column of a table is needed, the input vector should be set to (1,0,0...).

The projection is always following a restriction. For example, there is a query "select a, b from $Table$ where $(a + c) > 10$". In this way, after the restriction operation $(a + c) > 10$ is done, the projection operation picks up the attribute a and the attribute c from the tuples satisfying the restriction condition. This happens when the row multiplexing unit works, as shown in Fig. 6(a).

D. Aggregation

Among aggregation operations, the SUM() function is supported at present. A SUM() function accumulates the values of the same attribute in a set of tuples. SUM() can also be seen as an intermediate result of function AVG() that computes average value. Utilizing the structured storage scheme, the RRAM crossbar can also support the calculation of SUM() aggregation using PIM characteristic.

As shown in Fig. 6(b), the column-wise PIM direction is used for SUM() aggregation. A SUM() function only uses a sub-set of tuples that are selected by the restriction step, and accumulates a specific attribute value of these tuples together. Therefore, we can input the same computing voltage to the rows if the corresponding tuple is selected, and input zero voltage to the rows that are not selected.

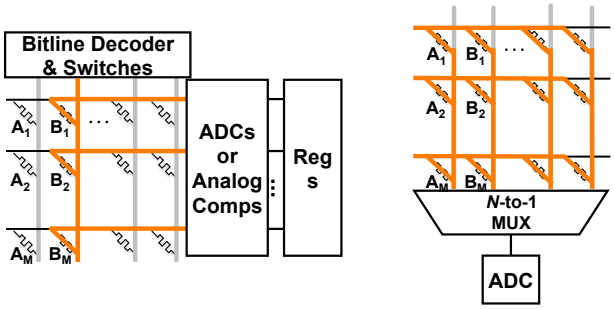

Fig. 6: (a) Projection operation implemented on RRAM crossbar. (b) Aggregation operation implemented on RRAM crossbar.

Based on the column-wise PIM structure of RRAM crossbar, the output current of each column reflects the results of SUM() for the corresponding attribute. Since the ADC overheads of energy and area are large and we can only operate row-wise computation or column-wise computation at a time, the ADC from the row ports in Fig. 6(a) and column ports in Fig. 6(b) can be reused.

V. RRAM-based SQL Query Unit

The structure of the proposed RRAM-based SQL query unit is shown in Fig. 7. Besides the PIM module described in Sec. IV, the modules to support entire SQL query and the I/O interfaces are contained in the unit.

A. Flow and Instructions of SQL Query

The function of a SQL query can be accomplished by four steps in the proposed RRAM-based SQL query unit: 1) restriction operations can be performed in PIM module; 2) restriction operations to be performed in peripheral module; 3) projection; and 4) aggregation. A query can contains only a part of steps. For example, the query "select $*$ from $Table$ where $(a + 3 * b) - (c + 2 * d) > 10$" only consists of steps 1) and 3).

We regard the four steps as four basic instructions for RRAM-based SQL query, and a controller is designed to change the data path according to the current instruction. We also need a Read instruction and a Write instruction to perform the memory function.

B. Result Buffer

Since a query contains multiple steps and the RRAM crossbar is re-used in steps 1), 3), and 4), one query needs to be accomplished by multiple cycles. In this way, the intermediate data need to be buffered for the next step. We use a result buffer consisting of registers to store the intermediate data and output data coming from every step.

C. Peripheral Restriction Module

RRAM-based PIM structure can perform the linear-combinational restrictions with high energy efficiency, as described in Sec. IV-B. However, the other restrictions in SQL query, like the boolean restrictions, is difficult to be directly accomplished by the PIM structure of RRAM. Therefore, a peripheral restriction module is designed to support these operations outside the RRAM crossbar.

A peripheral restriction module consists of a boolean function unit, a digital comparator, and the necessary control circuits. The boolean function unit performs the AND, OR, NOT, XOR, NAND, and NOR operations in CMOS gates. The digital comparator is used to perform "=" operation in digital domain, because analog signal need two cycles to accomplish "=" operation. The inputs of the peripheral module come from the result buffer. After calculation, the outputs are stored back to the result buffer.

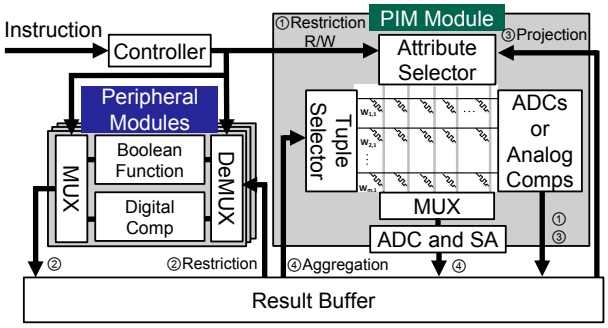

Fig. 7: Structure of RRAM-based SQL query unit.

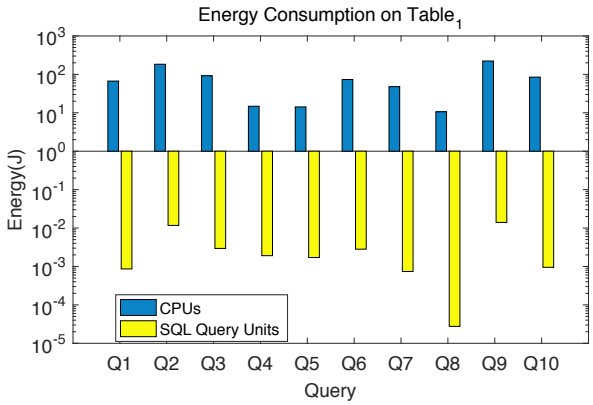

Fig. 8: Energy consumption for SQL unit evaluated in $Table_1$.

D. Creation, Update, and Deletion

Since we use RRAM crossbar as the database storage engine, the structured storage should also support the creation, update, and deletion operations in database applications. For creation operations, if a new attribute or a new tuple is generated, the corresponding value is stored in the last column or last row in the crossbar by default. For update operation, the address decoder selects corresponding position and then the write driver updates the value in the crossbar. For deletion operation, there is a tag with each tuple/attribute to show its valid state. When deleting a tuple/attribute, we turn valid tag from 1 to 0, and the corresponding row is available for creating a new tuple/attribute. The tags are maintained and stored outside this crossbar. In this way, the structured storage scheme can be maintained after creation, update, and deletion operations.

E. Split Large Table

A table in database applications can be very large, but the size of RRAM crossbar is limited by the fabrication technologies. Since a single RRAM crossbar may be not large enough to store an entire table, we need to split a large table into multiple parts and store each part in a RRAM crossbar separately. An intuitive method is to directly split a table. However, mapping several parts of a table into respective RRAM crossbar may lead to other overheads for query processing. We take the query "select $SUM(N)$ from $Table$ where $N + M < 10$" as an example to illustrate this problem. If the attribute M and N are stored in the same crossbar, the restriction $N+M < 10$ can be implemented by calculation on crossbars and an analog comparator, as shown in Sec. IV-B. However, if the attribute M and N are stored in two different crossbars, the data values in M and N should be converted from analog to digital, registered, and then accumulated for further comparison judgement. Another situation is that there are too many rows in a table to be stored in a crossbar. When calculating SUM() function for aggregation, each crossbar accomplishes the accumulation and the results are then merged together.

Due to the reasons above, we need to split a table into different crossbars while remain the query execution efficiency. In this work, we split a table following three principles. 1) When the table is cut by row, we just simply split the table. 2) When the table is cut by columns, we mark the order of each column and then store the strong-relative columns together as much as possible. The strong-relative columns means these columns are frequently mentioned together in restriction operations. We analyze the priori knowledge of possible queries and pick up those columns mentioned in the same queries in a same crossbar. 3) Once the the data location is manually determined at initialization time, it remains unchanged in the following query processing.

VI. Experiment Results

In this section, we evaluate the energy efficiency of the proposed RRAM-based SQL query unit. Since SQLite is a widely used embedded in-memory database engine, an in-memory database in SQLite is tested as the control group.

A. Set-up

We choose 10 representative queries and generate two test tables in database. One table contains 2^{20} tuples and 16 attributes named as $Table_1$. $Table_1$ is used to evaluate the situation without a table split by column. The other table contains 2^{15} tuples and 512 attributes named as $Table_2$. As contrast, $Table_2$ is used to evaluate the situation with a table split by column. We make a comparison of the energy consumption between directly table split and gathering relative columns. Both tables are 64 MB, in which the data type is 32-bits integer. Each table contains a primary key attribute id, and other 15 attributes $a, b, c, ..., n, o$ that are mentioned in the queries.

The queries we used in the experiments are as follows:

Q1: select id from $Table$ where $a < 30$;

Q2: select $*$ from $Table$ where $(c + h - a + o) < 60$;

Q3: select id, a, g, m from $Table$ where $(c + h - a + o) < 60$;

Q4: select $*$ from $Table$ where $(b + c + d) = 150$;

Q5: select id, b, c, d from $Table$ where $(b + c + d) = 150$;

Q6: select $*$ from $Table$ where $((e + f + g) + 1.2 * (h + i + j) + 0.8 * (k + l + m)) > 520$;

Q7: select id, e, h, k from $Table$ where $((e + f + g) + 1.2 * (h + i + j) + 0.8 * (k + l + m)) > 520$;

Q8: select $sum(m)$ as sum_m, $sum(n)$ as sum_n from $Table$ where $(m + n) < 30$;

Q9: select $*$ from $Table$ where $((a + b - c) > (d + e - f)$ AND $(h + i) < (j + k))$ OR $(c + d + e) < (k + l)$;

Q10: select id from $Table$ where $((a + b - c) > (d + e - f)$ AND $(h + i) < (j + k))$ OR $(c + d + e) < (k + l)$;

The control group in this work is an in-memory database in SQLite 3.16.0, running macOS sierra 10.12.4 on a MacBook Pro with 2 GHz Intel i7 core and 8 GB 1600 MHz DDR3. To measure the energy efficiency of these SQL queries on CPUs, we use the Intel Processor Counter Monitor (PCM) [22].

The crossbar array in each SQL query unit in this work is 256×256. We use a Verilog-A RRAM model to simulate the performance of RRAM-based design in SPICE [23]. We use existed design of DAC [24] and ADC [25] in the proposed unit. For other existed digital modules, we choose the same designs as those in the related work [7], [10], [26]. The customized circuits are modeled by Predictive Technology Model (PTM) [27].

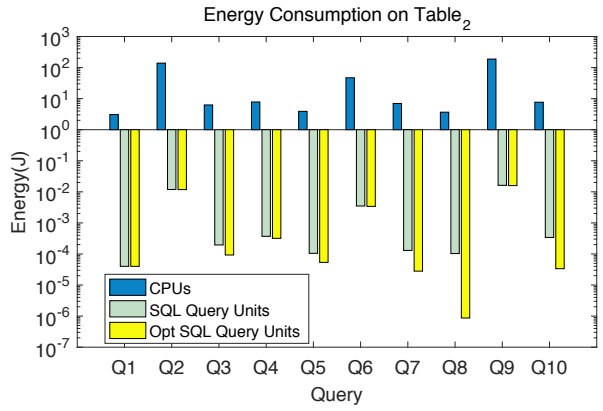

Fig. 9: Energy consumption for splitting $Table_2$.

B. Energy Evaluation

The energy consumption on $Table_1$ in Fig. 8 shows that the proposed SQL query units improve the energy efficiency by 4 to 5 orders of magnitude compared with CPUs. $Table_1$ does not need table split by columns, and it is split by rows when stored in the crossbars. The only affected query is Q8 in which there are SUM() functions. The extra cost for Q8 is to use ADC and registers to buffer the intermediate results from each unit when SUM() is calculated. Although there are external ADCs and registers used in Q8 here, the energy consumption in Q8 is still less than other queries. This is because the SUM() function can add most data in the crossbar by the column-wise computation and thus use fewer ADCs.

We run the queries mentioned above on the $table_1$ and $table_2$. The only difference is that the attributes mentioned in the queries, $id, a, b, c, ..., n, o$, are scattered over the 512 columns in $table_2$. The reason for not making these columns adjacent is that we want to show the comparison on the energy efficiency before and after we re-arrange the columns.

The energy consumption on $Table_2$ in Fig. 9 shows that the proposed SQL query units with directly table split improves the energy efficiency by 4 to 5 orders of magnitude compared with CPUs. After gathering the relative columns in the some crossbar, which is called optimized SQL query unit in Fig. 9, the energy consumption gets further reduced in some queries. In this way, the energy efficiency can be increased by 4 to 6 orders of magnitude.

The optimized split is effective especially for Q7, Q8, and Q10. There are two reasons. 1) It has been mentioned in Sec. V-E that the direct table-split method may lead to a great deal of extra ADC operations and other calculations. The burden can be alleviated if we put these columns that mentioned in the queries together in the same unit. 2) In Q7, Q8 and Q10, the volume of final results that return back from the SQL query units is small. The energy spent on reading final results are relatively small compared to the query processing. In this way, the energy efficiency gets a great improvement after the restriction operation is optimized.

VII. Conclusions

In this work, an RRAM-based SQL query unit has been proposed for high-energy-efficiency database applications. We have proposed an RRAM-based structured storage scheme which supports reading rows and columns from a table to avoid cache miss. An RRAM-based multi-mode SQL query unit has been proposed, which implements restriction, projection, and aggregation operations with PIM characteristic. For large database table, we have proposed a correlation-specific splitting method to store the table in multiple RRAM crossbars and to minimize the unit-to-unit communication. Simulation results have shown that the proposed unit obtains 4 to 6 orders of magnitude in energy efficiency compared with traditional architecture.

References

[1] Y.-C. Tu *et al.*, "A system for energy-efficient data management," *ACM SIGMOD Record*, vol. 43, no. 1, pp. 21–26, 2014.

[2] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE Pervasive computing*, vol. 3, no. 1, pp. 46–55, 2004.

[3] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st conference on Computing frontiers*, p. 162, ACM, 2004.

[4] H. Plattner and A. Zeier, *In-memory data management: technology and applications*. Springer Science & Business Media, 2012.

[5] H. Li and Y. Chen, *Nonvolatile Memory Design: Magnetic, Resistive, and Phase Change*. CRC Press, 2011.

[6] I. Jo *et al.*, "Yoursql: a high-performance database system leveraging in-storage computing," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 924–935, 2016.

[7] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, pp. 27–39, IEEE Press, 2016.

[8] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication," in *DAC*, pp. 1–6, IEEE, 2016.

[9] L. Gao, P.-Y. Chen, and S. Yu, "Demonstration of convolution kernel operation on resistive cross-point array," *IEEE Electron Device Letters*, vol. 37, no. 7, pp. 870–873, 2016.

[10] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, pp. 14–26, IEEE Press, 2016.

[11] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," HPCA, 2017.

[12] P. Bakkum and K. Skadron, "Accelerating sql database operations on a gpu with cuda," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pp. 94–103, ACM, 2010.

[13] J. Casper and K. Olukotun, "Hardware acceleration of database operations," in *FPGA*, pp. 151–160, ACM, 2014.

[14] D. Ziener *et al.*, "Fpga-based dynamically reconfigurable sql query processing," *ACM Trans. RETS*, vol. 9, no. 4, p. 25, 2016.

[15] C. Dennl, D. Ziener, and J. Teich, "On-the-fly composition of fpga-based sql query accelerators using a partially reconfigurable module library," in *FCCM*, pp. 45–52, IEEE, 2012.

[16] C. Dennl, D. Ziener, and J. Teich, "Acceleration of sql restrictions and aggregations through fpga-based dynamic partial reconfiguration," in *FCCM*, pp. 25–28, IEEE, 2013.

[17] L. Wu *et al.*, "The q100 database processing unit," *IEEE Micro*, vol. 35, no. 3, pp. 34–46, 2015.

[18] S. D. Viglas, "Data management in non-volatile memory," in *SIGMOD*, pp. 1707–1711, ACM, 2015.

[19] A. Chatzistergiou *et al.*, "Rewind: Recovery write-ahead system for in-memory non-volatile data-structures," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 497–508, 2015.

[20] H. Kimura, "Foedus: Oltp engine for a thousand cores and nvram," in *SIGMOD*, pp. 691–706, ACM, 2015.

[21] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, "Sneak-path testing of crossbar-based nonvolatile random access memories," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 413–426, 2013.

[22] "Inter pcm." https://software.intel.com/en-us/articles/intel-performance-counter-monitor/.

[23] S. Yu *et al.*, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Advanced Materials*, vol. 25, no. 12, pp. 1774–1779, 2013.

[24] S. Y.-S. Chen, N.-S. Kim, and J. Rabaey, "A 10b 600MS/s multi-mode CMOS DAC for multiple Nyquist zone operation," in *Symposium on VLSI Circuits*, pp. 66–67, IEEE, 2011.

[25] M. J. Kramer *et al.*, "A 14 b 35 MS/s SAR ADC achieving 75 dB SNDR and 99 dB SFDR with loop-embedded input buffer in 40 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 12, pp. 2891–2900, 2015.

[26] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," pp. 15–50, Springer, 2014.

[27] W. Zhao and Y. Cao, "Predictive technology model for nano-cmos design exploration," *ACM JETC*, vol. 3, no. 1, p. 1, 2007.