This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



# NVQuery: Efficient Query Processing in Non-Volatile Memory

Mohsen Imani, *Student Member, IEEE,* Saransh Gupta, *Student Member, IEEE,*
Sahil Sharma, and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Today's computing systems use huge amount of energy and time to process basic queries in database. A large part of it is spent in data movement between the memory and processing cores, owing to the limited cache capacity and memory bandwidth of traditional computers. In this paper, we propose a non-volatile memory-based query accelerator, called NVQuery, which performs several basic query functions in memory including aggregation, prediction, bit-wise operations, join operations, as well as exact and nearest distance search queries. NVQuery is implemented on a content addressable memory (CAM) and exploits the analog characteristic of non-volatile memory in order to enable in-memory processing. To implement nearest distance search in memory, we introduce a novel bitline driving scheme to give weights to the indices of the bits during the search operation. To further improve the energy efficiency, our design supports configurable approximation by adaptively putting memory blocks under voltage overscaling. Our experimental evaluation shows that, NVQuery can provide 49.3× performance speedup and 32.9× energy savings as compared to running the same query on traditional processor. Approximation improves the energy-delay product of NVQuery by 7.3×, while providing acceptable accuracy. In addition, NVQuery can achieve 30.1× energy-delay product improvement as compared to the state-of-the-art query accelerators.**

***Index Terms*—Query processing, Non-volatile memory, In-memory computing, Content addressable memory**

## I. Introduction

Data management systems (DMS) are the standard tools for collecting and serving large amounts of information for web applications and end users. Over the past decade, data generation has grown exponentially due the diversity of collection sources [1]–[3]. In addition, organizations collect large amounts of information for decision making and business analytics [4]–[6]. In the majority of scenarios, the execution time of DMS queries tends to increase linearly and sometimes exponentially as more records are stored in a single server instance. This has been one of the main challenges of DMS and its caused by the the hardware and software co-design limitations [7].

Several efforts have been made to accelerate computation by paralleling operations on a co-processor or GPUs [8]–[14]. However, in most cases data movement has been a bottleneck due to the fact that large amounts of information tend to reside in memory. In most Structured Query Language (SQL) accelerator studies, this data overhead is not taken into account and as a consequence their results do not show a valuable improvement over general designs [15]. On the other hand, software have been developed to adapt to the nature of particular tasks. In the case of interactive data analysis, the focus is often less on exactness of the result and more on timeliness or responsiveness, which gives us the opportunity to approximate the result within a margin of error [7]. Data movement is the main bottleneck of current computing systems wherein the size of data increases over the cache capacity of the processing core [16]. Limited memory bandwidth makes the condition worse as data is delayed each time the main memory is accessed.

Near data computing and processing in-memory (PIM) are two efficient techniques which reduce the cost of data movement [17]–[23]. Near-data computing puts the computing units close to the main memory, in order to avoid data movement cost in computation [24], [25]. Although this technique improves the computation efficiency, it has some challenges including: (i) cost of large CMOS-based computing unit and (ii) cost of integrating the memory and logic in a single chip. The introduction of non-volatile memories has made it possible to process data in the memory itself [26]–[32], resulting in the concept of PIM [17], [19], [33]. Resistive RAM (ReRAM) is one such memory, which enjoys the benefit of low energy, high switching speeds, high density, and scalability. PIM processes data within memory, eliminating the need for integration between large processing cores and the memory. However, the existing PIM techniques support only simple functions like bit-wise or search operations [34], [35]. Not only it is too cumbersome to break down a simple query function like search into a series of bit-wise computations but it also minimizes the benefits of using PIM.

This paper implements an efficient PIM-based query processor which supports a wide range of query functions. We propose a novel non-volatile, memory-based query processing accelerator, called NVQuery. NVQuery supports wide range of query functionalities including aggregation functions, prediction functions, bit-wise operations, addition, joins, exact and nearest distance search operation. The configurable crossbar memory structure of our design supports these functionalities inside the memory. It exploits the analog characteristic of non-volatile memory to also enable the nearest distance search capability. The exact search mode enables NVQuery to realize join operations. Our experimental evaluation shows that, NVQuery can provide 49.3× performance speedup and 32.9× energy savings as compared to running the same

M.Imani, S. Gupta, S. Sharma and T. Rosing are with the department of computer science and engineering, University of California San Diego, La Jolla, CA, 92093.
E-mail:{moimani, sgupta, sas110, tajana}@ucsd.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



query on traditional processor. Approximation improves the energy-delay product of NVQuery by 7.3×, while providing acceptable accuracy. In addition, NVQuery can achieve 30.1× energy-delay product improvement as compared to the state-of-the-art query accelerators.

## II. Related Work

### A. Query Processing

Several efforts have been made in order to accelerate DMS querying by using specialized hardware [8], [36]–[38]. GPUs in particular have been used to parallelize the 'SELECT' SQL queries with results that range from 20x to 70x speed up [8]. However, they do not take into consideration the data movement overhead of these tasks and assume only the computation cost. It has been demonstrated that the bandwidth and cache capacity of GPU devices are the main bottlenecks of database computations. For instance, work in [36] examines multiple GPU systems and acknowledges that unless the full working set of data can fit into the memory on a GPU, PCI Express bus will be a bottleneck.

As a consequence, researchers have worked on optimizing the data movement through memory. In the areas of distributed computation, Al-Kiswany *et al.* describe StoreGPU, a distributed storage system that uses pinned, non-pageable memory on the host system to reduce the impact of data transfer [39]. Gelado *et al.* in [40] introduce an asymmetric distributed shared memory that defines two types of memory updates which determine when to move data on and off the GPU. These optimization efforts only focus on conventional memory technologies. The computation still occurs on computing units. A query service by Google called, BigQuery is capable of searching through petabytes of data [41]. The latency is minimized by paralleling queries over multiple servers and columnar storage of data over multiple memories or memory banks. However, this distribution of queries and data results in huge energy requirements. Also, it does not support data manipulation queries and has large latency overhead for updating data.

On the other hand, the work in [42] uses analog characteristic of non-volatile memory and analog to digital converter (ADC) to enable query processing in-memory. However, prior work showed that ADC are taking large portion of memory energy and area [19]. Also, it utilizes multi-level memristors which are highly unreliable and restrict the size of numbers that can be stored in each memory cell. Each attribute in the paper is a 32-bit integer, which is practically not feasible with the type of RRAM device used.

Our paper adds support for a wide range of functions to extend the capabilities of NVQuery. We further introduce approximation techniques to improve the performance and energy efficiency of NVQuery.

### B. SQL Optimization

Approximating the results of SQL query can be used to reduce the required waiting time by producing results within acceptable error bounds. The most famous querying framework based on approximation is sampling-based approximate querying (SAQ) [43], [44], where the computation is performed over a small random subset of the data. The error in the estimate is specified using a confidence interval or error bars. However, SAQ ignores the tails of the data that cannot help with complex queries. Poti *et al.* [7] proposes deterministic approximate querying (DAQ) schemes that formalize a deterministic approach to approximate the results by taking advantage of the bit value representations. Their approach reads the table records, starting from the most significant bit, one by one and adjust deterministic error bounds with respect to the bits not seen yet. DAQ evaluations estimates less than 1% error with a speedup of 6x for SQL predicate queries and 2-4x for aggregation.

SQL supports many operations to make the processing of queries easier. Out of these, join is considered to be one of the most important operation. Join splits data into tables, saving memory space by removing the redundancy associated with one monolithic table. Thus, it is one of the most basic operation, with considerable cost because of the sheer amount of data being read from storage. The efficiency of join operation is one of the most fundamental concepts of relational databases. Recent years have seen many papers that deal with the worst case scenario speedup for joins [45], [46]. Innovative techniques for different distributed systems [4], [47] show the need to improve the efficiency of join operation in databases.

Software based optimization for joins using dynamic and greedy algorithms have looked into SMP based multi-joins [48]. While this tackles the problem by building an analytical model, it does not optimize the core join operation for the general use case. Join optimizations for MapReduce environment give promising results for some cases of big data joins [49], [50]. It tries to reduce the communication, but still suffers from the lack of core join optimization. NVQuery can potentially be used to optimize and accelerate all such methods as it works directly in memory.

## III. NVQuery Accelerator

Fig. 1 shows the general architecture of the proposed NVQuery. The proposed NVQuery integrates with DRAM and enables the main processor to accelerate query processing. NVQuery can also be used as a secondary storage to improve the effective DRAM capacity. NVQuery consists of $N$ banks, where each has $k$ memory blocks. Each memory block can be configured as memory or query accelerator.

Our design is a heterogeneous architecture, where the NVQuery co-operates with main processor in order to find the query result. In NVQuery, each memory block returns a result of the query, independent from other blocks. Therefore, to find the result of a query from the whole data set, the main processor receives output response of each memory block (a total of $N \times k$ values instead of the entire data). Finally, it processes data to find the result of query over the entire data set. In this way, the load on memory bandwidth due to query processing and its related costs are significantly reduced. Table I lists different configurations that NVQuery can take including: nearest search, search, and memory. For each of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



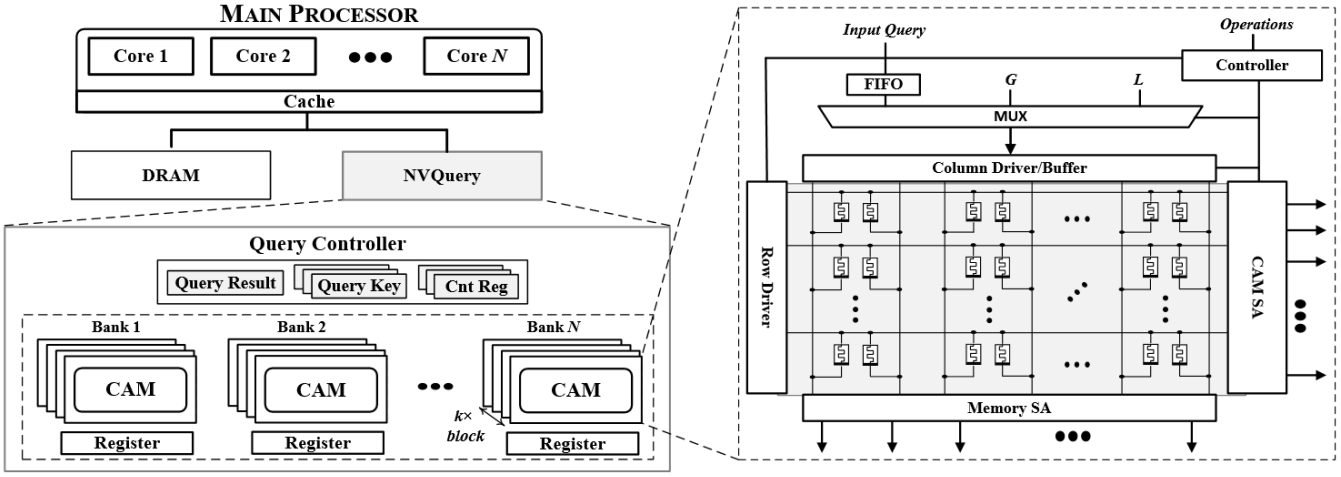


Fig. 1. Proposed NVQuery architecture with $N$ banks and $k \times N$ blocks. The right part details the crossbar implementation of memory banks along with the supporting control logic.

TABLE I
NVQUERY SUPPORTED CONFIGURATIONS

| Configuration | Example Functions | CAM Input | CAM SA | Memory SA | Comment |
|---|---|---|---|---|---|
| **Nearest Search** | MIN | Least | Nearest | NA | L: Least possible value |
| | MAX | Greatest | Nearest | NA | G: Greatest possible value |
| | TOP K | IQ (FIFO) | Nearest | NA | Requires k iterations |
| **Search** | Exact Search | IQ (FIFO) | Exact | NA | IQ: Input query |
| **Memory** | Bitwise | NA | NA | AND/OR | CAM input: Bitline driver |
| | Memory | NA | NA | MEM | CAM input: Bitline driver |
| | Addition | NA | NA | MAJ | CAM input: Bitline driver |

TABLE II
NVQUERY SUPPORTED FUNCTIONALITY

| | Notation | Functions |
|---|---|---|
| Aggregation | $F(S_I) \rightarrow S_O$ | MIN, MAX, Average, Count |
| Bit-wise Operations | $F(S_I) \rightarrow S_O$ | AND, OR, XOR (Combination of AND, OR) |
| Addition | $F(S_I) \rightarrow S_O$ | In-memory addition |
| Comparison | $= \ \leq \ \geq$ | Bit-wise and value-wise comparison |
| Predict | $p$ | Exist, Search condition, Top, Like, Group, Between |
| Join | $\bowtie, \rtimes, \ltimes, \rtimes\ltimes \ldots$ | Inner, Left, Right, Outer, Semi joins |

the configurations, we show the status of different memory peripherals for some example functions. In this section, we describe the functions supported by our proposed non-volatile query processor, NVQuery. Table II lists the NVQuery support functionalities. NVQuery supports a large number of essential functions including aggregation (`MIN`, `MAX`, `Average`, `SUM`, and `Count`), boolean functions (such as `AND`, `OR`), addition, comparison (equality or non-equality), and different types of Join. In addition, NVQuery can also process prediction functions such as `Exist`, `Search Condition`, `Like`, `Group`, `Between`, and `Top` in memory.

We map all query functionalities explained in Table I to NVQuery which can work in three main configurations: (i) look-up table (LUT) with capability of exact search, (ii) nearest distance search, and (iii) memory. We propose a new memory architecture which can process data locally without reading it. In each of these configurations, our design shown in Fig. 1 processes query operations without approximating the result. In the following subsections, we explain how each query operation can be supported in memory.

### A. Exact Search

The most common operation in many query processors is looking up for a set of data which matches with input query. A typical search query involves a brute-force search through a LUT till the data is located. This is usually implemented in one of the two ways, (i) word-by-word search and (ii) bit-by-bit search. A word-by-word search looks through every stored word in the LUT sequentially and finds a match. In the worst case, it involves processing each and every element present in the LUT. The bit-by-bit search scans through one bit (but same index) for multiple words at a time. The first iteration analyses a particular bit index of every word in the LUT, looking for a match with the corresponding entry in the input query. The following iterations are performed only on the words filtered by previous iterations. This approach does not analyze all the elements since the size of candidate pool decreases after each iteration. The exact search operation supports the following functions in NVQuery:

`Exist`: It is used to test the existence of some specific data in the LUT. The exact search can be directly used to implement this function.

`Count`: It is used to get the number of rows that match a certain criteria. The `Count` output can be obtained by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



counting the number of hits for an exact search query. Our design adds a counter block to NVQuery in order to support this query.

`Like`: It is used to find the existence of a specific data or pattern of data in the rows. It involves searching for occurrence at (i) a particular position and (ii) any position. The first case can be easily implemented using the exact search mode in the same way as the `Exist` function. The second case requires repeated use of exact search mode for all the occurrence patterns possible. This comes with an inherent latency overhead due to multiple serial exact searches.

`Group by`: It is used to group the rows on the basis of one or more columns. The grouping is usually based on the output of some operation applied to the data in the column. Multiple serial exact searches are used to find the rows belonging to different groups.

### B. Nearest Distance Search

NVQuery can be configured to perform the closest distance search operation inside the memory. The bit-by-bit search described above can be used to implement this functionality. Here, the nearest data is the one which remains selected for the maximum number of iterations. Our design exploits this functionality to support aggregation functions like `MIN` and `MAX` and prediction functions like `Top k`. Running these queries on traditional core has a time complexity of $O(\log n)$. However, our hardware can find `MIN, MAX` queries in a single cycle and `Top k` in k cycles.

`MIN`: This query runs on a set of stored data to find the minimum value. To perform this query in LUT, NVQuery block adopts the nearest distance search configuration and searches for the data which has the closest distance to the minimum possible value. In case of unsigned numbers, our design searches for an entry which has the closest distance to zero. In the case of signed values, this number is the largest possible negative number (single one followed by a chain of zeros, i.e., 1000...0).

`MAX`: To find the data with the maximum value, we search for the entry which has the least distance from the largest positive number. For unsigned values, the largest value is a chain of ones (1111...1), while in the case of signed numbers, this value is represented by a zero followed by a chain of ones (0111...1).

`Top k`: To search for $k$ values closest to the input data, we perform the nearest distance search for $k$ iterations. After each iteration, our design deactivates the selected word and repeats the nearest distance search on the remaining words. This approach gives a set of $k$ nearest values arranged in the order of their proximity to the input. Our design also supports bit-wise/value-wise comparison by searching for the exact and nearest values.

`Between`: This operator takes in two inputs, the lower and upper limits, and outputs those values from the stored data which are equal to or between these limits. Traditional implementations of this function involve a lot of computational overhead, comparing each value with the limits. The nearest distance search proposed above enables efficient implementation of this operator. Instead of finding the values nearest to the upper and lower limits, we find the values nearest to the midpoint of the total range. Then the values are sorted as explained in Section IV-B. NVQuery compares the values with the input limits and selects the entries which lie between them. Instead of naively searching through the data, NVQuery uses binary search to find the corner cases and reduce the computational overhead.

### C. Join

NVQuery supports different types of joins namely, `inner, left, and right joins`. Our implementation is similar to in-memory hash joins, but more efficient due to NVM-based PIM. Ideal implementation of join would involve fetching the data from memory to the core and searching through the involved tables. Although, optimizations like hash join reduce the amount of data to be transferred yet the cost of data movement is a lot. NVQuery reduces this overhead by reducing searching for keys inside the memory itself. The exact search discussed earlier is used to implement joins.

*Equi joins* involve searching for exact match of the join key through the tables. Exact search mode can be easily extended to implement different kinds of *equi* joins, enabling the records that are needed for the final join computation. Memory read bus along the columns of a table is used to read the desired columns of the rows with matching data. The read data is copied to the memory location pertaining to the final join output. Limited 4K row capacity forces the implementation to break the table into multiple 4K slices or blocks, this does not affect the computational complexity of the implementation and the impact on execution time is also minimal. A choice between a block or slice is made based on the query and size of the input.

NVQuery is flexible enough to cater to any combination of columns to implement different types of joins like inner, left, and right joins. Multi joins are implemented by saving the temporary result of two table joins and iteratively applying joins to that.

Different SQL implementations use a combination of nested loop, merge and hash joins. Execution complexity of these methods vary based on availability of index on the join property. The worst case complexities are $O(NM), O(MlogN+NlogM)$ and $O(N+M)$ respectively, where *N,M* are the table sizes. NVQuery's worst case execution complexity is equivalent to hash joins, though it does not require explicit hashing of the join key like hash joins do.

### D. Bit-wise Operations and Addition

A traditional processor implements bit-wise logic operations in the main core. The operands are fetched from the main memory and brought through the memory hierarchy all the way up to the core. The core then performs the required computations. On the other hand, our design implements these operations in the memory, avoiding the need to transfer data from memory. For executing these operations, NVQuery is set into memory configuration and the output is obtained from memory SA. This operation can support the following queries: `AND, OR, XOR` and `Average`. Our design supports

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



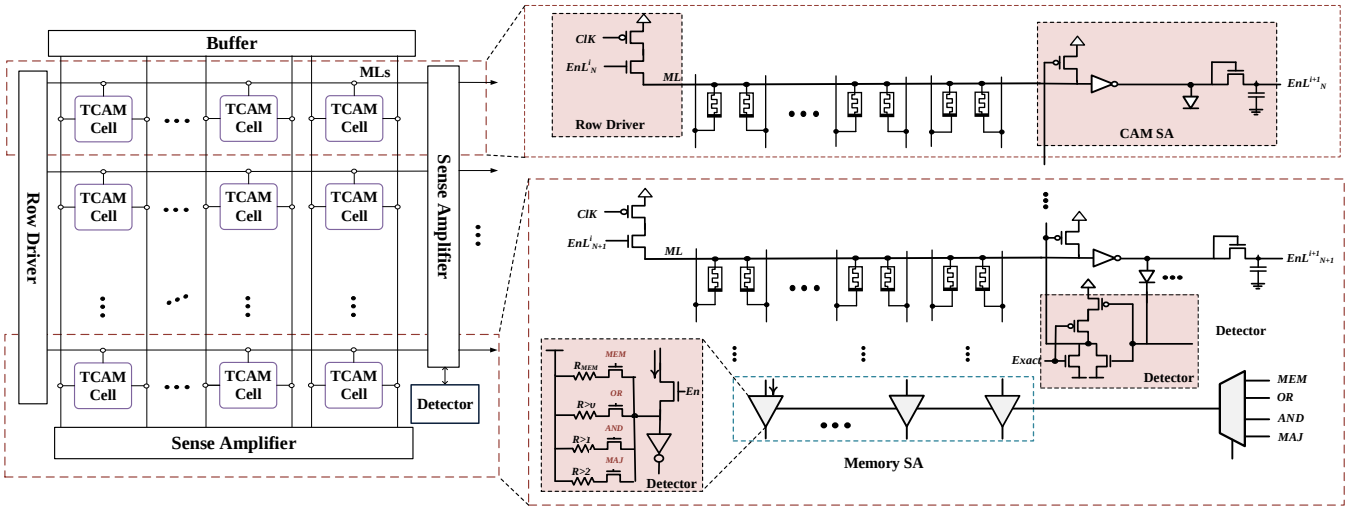


Fig. 2. Circuit level implementation of CAM SA, Memory SA, and Row Driver.

average query by using a counter and sending the data to main processor.

## IV. Hardware Support

This section describes the hardware implementation of NVQuery and the way in which it supports the functions described in Section III. NVQuery is designed using a crossbar non-volatile memory architecture. The crossbar is configured in such a way that a set of two storage elements in the crossbar corresponds to one bit data. Data 0 is stored as $\{R_{HIGH}\ R_{LOW}\}$, while 1 is stored as $\{R_{LOW}\ R_{HIGH}\}$. However, our architecture does not use any access transistors for these elements, hence it is called 0T-2R. Implementations like 2T-2R require access transistors. This makes the design unsuitable for a crossbar memory, reducing the area density benefit of non-volatile memories. Moreover, the presence of transistors introduces non-linearity to the system. On the other hand, 0T-2R doesn't need access transistors and can be implemented on a conventional crossbar memory, making it more area efficient.

As shown in Fig. 1, the crossbar memory in NVQuery is supported by peripheral components. The controller receives the input query and generates the appropriate control signals. It is also responsible for collecting the output of the block and forwarding it for further processing. The multiplexer managed by the controller, selects the input which drives the bitlines of the crossbar memory. This input can either be the input query (in case of search operations) or greatest positive value (corresponding to `MAX`) or least representable value (corresponding to `MIN`). The column driver drives the bitlines of the crossbar. It not only applies the execution voltages for different operations but also maps the input query to the required bitline voltage levels. Row driver is responsible for charging the wordlines (also called match-lines due to the nature of operations). It is also responsible for selecting/activating different words (rows) in the memory. It also provides a limited set of voltage options essential to the working of crossbar. The crossbar is equipped with sense amplifiers (SAs) on both the wordlines (CAM SA) and the bitlines (memory SA). Fig. 2 shows these SAs. The CAM SAs are responsible for detecting charging and discharging behavior of wordlines. The nMOS-capacitor circuit acts as a latch. The inverter-diode-NOR circuit deactivates the wordlines as soon as the first edge is detected or the sampling signal for *Exact* is set. As a result, the latch is set only for the wordlines which discharge before this deactivation. The memory SAs are buffers with special resistors to support bit-wise and memory operations as described in Section IV-C. We next discuss how NVQuery enables different functions discussed in Section III.

### A. Exact Search

To implement the LUTs discussed in Section III-A, NVQuery uses content addressable memory (CAM) configuration of crossbar. Fig. 2 shows the structure of non-volatile crossbar CAM, capable of searching for stored data which exactly matches the input query. During search operation, all the match-lines (MLs) pre-charge to $V_{dd}$. The input buffer (column driver) distributes the query point to all CAM rows using vertical bitline. Any cell with the same stored data as input query discharges the ML. The sense amplifier, connected to the horizontal ML, determines the equality of the input and stored data by sampling the ML voltage [51].

Consider a data set which contains the name, age, height, and income of people in different companies. The query `SELECT F(income) FROM COMPANY1` is an example of SQL query. For this query, a query processor first selects all people in the list which are working for the `COMPANY1`. Then it applies another query function, `F`, on the `income` of all selected people. NVQuery eliminates the need for multiple sequential searches. It can perform a single step search by activating the bitlines corresponding to `COMPANY1` and `income` simultaneously. The output of the query is given by the rows with fastest discharging MLs. This not only saves time by eliminating multiple searches but also the power involved in repeated charging and discharging of MLs. Each memory block/LUT returns an output to the controller. Finally,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



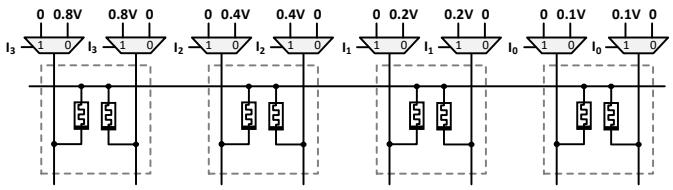


Fig. 3. NVQuery in nearest distance search configuration.

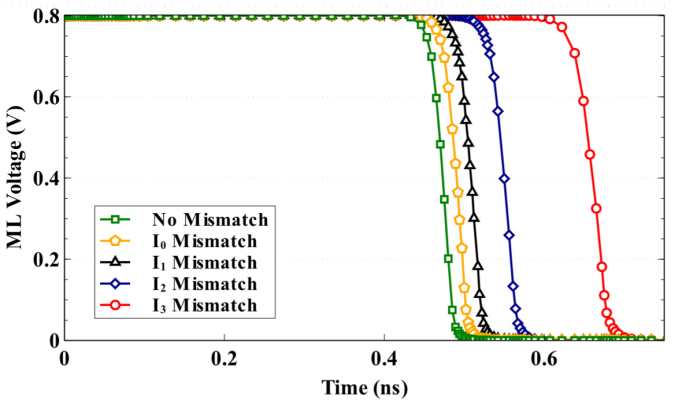


Fig. 4. Timing characteristic of CAM block in nearest distance search configurations.

the output data from each block is processed by the main processor which evaluates the final query result.

### B. Nearest Distance Search

CAM has been extensively used to implement search operations. Different versions of CAM implementations (*e.g.* TCAM) on different types of hardware (crossbar, 2T-2R, 3T-1R, *etc.*) have been active topics of research recently. However, majority of the previous work revolves around exact and nearest hamming distance search operations. Hamming distance is a good criterion when considering hyper-dimensional vectors where the index of a bit does not matter. Only the face value of a bit and the total number of mismatches between the stored data and the input query are considered. Such a comparison is not practical for many real life applications where a query to the processor is dependent on the binary weighted values of the stored data.

To support such queries, some researchers have proposed the division of a memory block into stages [52]. In such an architecture, the first $m$ most significant bits of data are stored in the first stage, the next $m$ significant bits in the second stage and so on. Then, a search is performed sequentially, starting from the first stage. The output of a stage selects the rows to be activated in the following stage. This increases the weight of the initial stages with respect to the later stages. However, the $m$ bits in a stage are treated as having the same binary weight. This leads to inaccurate results in many cases. In this work, we address this issue by introducing a new method to assign binary weights to the bits within a stage.

For a search in conventional CAM, the match-lines (MLs) are pre-charged to $V_{dd}$ and then bitlines are driven with $V_{dd}$ or 0 depending upon the input query. The MLs of rows with more number of matches discharge earlier. The line to discharge first is the one with minimum mismatch with the input query. To give binary weight to the bits, we modify the bitline driving voltage. Suppose a stage contains $m$ bits $(m-1:0)$. The bitlines which were earlier driven with $V_{dd}$ and now driven with a voltage $V_i = V_{dd}/2^{(m-1-i)}$ where $i$ denotes the index of a bit in the stage. Fig. 3 shows CAM in nearest search configuration for a stage size of 4 bits. As shown in Fig. 4, a match in the most significant bit results in faster ML discharging current than lower indices. We exploit this difference and design a CAM which can find the binary value nearest to the input query.

The different discharging currents also allow us to sort the data, with the nearest data discharging first. This sorting is easy for a smaller number of records. However, as the number of records increase, it becomes difficult to differentiate data depending upon the discharging currents. In such a case, nearest search is implemented in groups with limited rows selected at a time.

Now, as the number of bits increases, the bitline voltage $V_i$ becomes very small. We limit the minimum available voltage source output to $100mV$. Moreover, the maximum voltage that can be applied is limited by the threshold voltage of the non-volatile elements. This ensures that the data in the memory is preserved. This upper bound is set to $1.8V$. Hence, the allowable voltage levels include $0.1V, 0.2V, 0.4V, 0.8V$ and $1.6V$, restricting the stage size to 5 bits. In this work, we split the CAM into multiple stages of 4-bits each for simplicity and then search for the nearest distance row in a serial manner, starting with the stage containing the most significant bits.

### C. Bit-wise Operation and Addition

Although a search based CAM can accelerate several functionalities in NVQuery, it cannot support a major part of queries such as addition, average, and all bit-wise operations. In order to make NVQuery a general design for query processing accelerator, we modify the sense amplifiers in the vertical bitlines to support bit-wise operations. Fig. 2 shows the sense amplifier in a single NVQuery bitline to support bit-wise operations. In this mode, each block works as memory instead of CAM, where one of the vertical bitlines in each CAM cell is activated. The tail of the shared bit-line is connected to a sense amplifier. Since our design supports AND and OR functions, the sense amplifier has two main parts: one for AND operation and a simple sense amplifier to support OR. These circuits work on the basis of the leakage current through the vertical bitline. When several rows in memory are active, each row leaks current through vertical bitlines depending upon the resistance value. If the stored bit is 1 (low resistance), this current is large, while in the case of 0, leakage is significantly small. The goal of OR operation is to identify the presence of at least one high (1) bit in all activated rows. Therefore, we use a sense resistor, $R_{>}0$, such that in the case of at least single high bit, it turns the output signal to one. However, for AND operation the goal is to find a case such that at least one input is not 1. In that case, the AND circuitry uses an appropriate sense resistance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



TABLE III
APPROXIMATION IN 16-BIT ADDITION

| Approximated Bits | 4 | 8 | 12 | 14 | 16 |
|---|---|---|---|---|---|
| Error (%) | 0.006 | 0.098 | 1.56 | 6.25 | 25 |
| Energy (pJ) | 3.52 | 2.41 | 1.3 | 0.75 | 0.197 |
| Latency (ns) | 182 | 133 | 84.7 | 60.5 | 36.3 |

TABLE IV
NVQUERY APPROXIMATION AT DIFFERENT SUPPLY VOLTAGES

| Voltage | 1V | 0.87V | 0.8V | 0.74V | 0.7V | 0.67V |
|---|---|---|---|---|---|---|
| Errors bits | 0 | 1 | 2 | 3 | 4 | 5 |
| Norm. Energy | 1 | 0.68 | 0.39 | 0.22 | 0.17 | 0.11 |

Interestingly, prior work shows that crossbar memory can further support addition within the memory [53], [54]. This approach breaks down an operation into a series of NOR operations. The logic family used in the paper executes NOR in crossbar memory with a latency of just 1 cycle. This functionality is supported by NVQuery due to its regular structure (unlike CAMs with access transistors), enabling it to perform data computations within memory. In the case when approximate results are acceptable, the sense amplifier at the bitlines can be used to improve the performance of NVQuery. The truth table for 1-bit full adder shows that the sum bit (S) can be obtained by inversion of the carry bit (C) in 75% of the cases. The sense amplifier calculates C (majority) in one step by simply using an appropriate sense resistance. S is obtained by inverting C. This introduces a worst case error of 25%. However, this error is reduced significantly by approximating only some LSBs depending upon the level of accuracy desired. The MSBs are calculated accurately using the techniques described in [53]. Table III shows the error corresponding to different number of approximated bits for an 8-bit addition. By calculating the carry bit correctly, the proposed approximation approach limits the effect of an error to one bit and does not propagate it.

Addition is extended to implement average function. The output of successive additions is sent to the processor, where the average is obtained by bit-shifting or simple division.

## V. APPROXIMATION IN NVQUERY

In most cases, a query does not require a unique or completely precise answer. Instead, it requires a fast result with good enough accuracy. Approximate computing is an effective way of improving the energy and performance by trading some accuracy. Much of the prior work seeks to exploit this fact in order to build faster and more energy efficient systems which are capable of responding to our needs with just good enough quality of response [52], [55], [56]. However, most of the existing techniques provide less energy or performance efficiency due to considerable data movement and lack of configurable accuracy.

NVQuery can work in both exact and approximate mode. Approximate mode provides the advantage of better metrics, both in terms of latency and power consumption. However, this comes at the cost of loss in accuracy. Here, we investigate two ways of approximation: (i) bit trimming and (ii) voltage scaling.

### A. Bit Trimming

One common way to apply approximation in query search is trimming or neglecting bits. Our design neglects few least significant bits of input data in order to accelerate the query functionality. For other bits, NVQuery performs the search serially on the blocks, starting from the most significant bits. The level of approximation is tuned by determining the number of neglected blocks. The upper and lower computation bounds are defined by the number of cut bits. For each input in query, the lower bound is defined by all trimmed bits being zero while the upper bound by all trimmed bits being one.

$$L_V < V < U_V$$

$$U_V - L_V = 2^K - 1$$

Where $V$ is the exact value of $V$, and $L_V$ and $U_V$ are the lower and upper bounds respectively when the last $k$ bits are trimmed. Therefore, our design guarantees that the NVQuery error rate on aggregation functions, (Minimum, Maximum, Average, Mean, etc.) is

$$Error_{Query} < 2^{M-K} - 1$$

where $M$ is the total number of bits.

For a 5-bit CAM stage with a nominal $V_{dd}$ of $1.6V$, $V_i = \{0.1V, 0.2V, 0.4V, 0.8V, 1.6V\}$ for $i = \{0,1,2,3,4\}$. This leads to an effective difference of $\{1.5V, 1.4V, 1.2V, 0.8V, 0V\}$ between ML and the bitline. If the lower bits in a block are approximated to have the same weight, then the required number of voltage levels can be reduced. However, the voltage levels for the non-approximated bits should be chosen such that

$$V_i = \begin{cases} (k+1) \times 0.1V, & i = k \\ 2 \times V_i, & i > k \end{cases} \quad (1)$$

where k is the number of approximated bits. This ensures that the effective weight of the approximated bits is at least 1 LSB ($0.1V$) less than the first accurate significant bit. For example, if the lower 2 bits are approximated to have the same weight, then the required voltages are $\{0.1V, 0.1V, 0.3V, 0.6V, 1.2V\}$. This further reduces the required $V_{dd}$ for ML, reducing the total energy requirement of the computation.

### B. Voltage Scaling

In NVQuery, approximation is done by applying voltage overscaling (VoS) on selective CAM blocks [14]. While CAM works without any error with nominal $V_{dd}$, lower supply voltages increase the possibility of error on CAM matching and memory functionality. Table IV lists the possible errors for each CAM block at different supply voltages. For instance, a 6-bit CAM block at 870mV supply voltage can match the input query with stored data with a single bit mismatch. Similarly, at 800mV and 740mV, the CAM block can search for data with 2-bit and 3-bit Hamming distance respectively from the input key.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



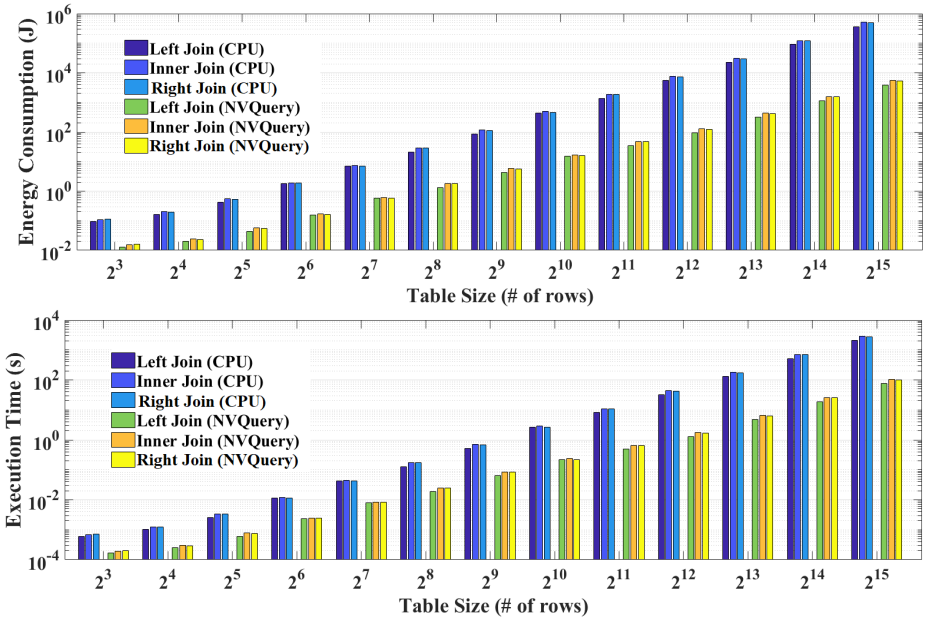


Fig. 5. Energy consumption and performance of running join operations with different table sizes on traditional cores and the proposed NVQuery.

Our design puts the blocks in different approximation levels based on their impact on approximation. For instance, if the $i^{th}$ block is configured with $h$-bit error, the $i-1^{th}$ block needs to have $h/2$-bit error. Generally, when the goal is to allow $K$ bit error, we can estimate the error distance of each bit as follows:

$$h = K/(1+1/2+1/4+\ \ldots\ +1/2^N)$$

Comparing these two ways of applying approximation shows that voltage overscaling can provide much higher advantage as compared to bit ignoring. In bit ignoring, the energy saving and speedup limits to a few bits which we neglected processing them. For example, in 6-bit CAM, trimming 2-bit, will give us $2/6 = 33\%$ energy savings.

## VI. Experimental Results

### A. Experimental setup

For detailed evaluation of the proposed NVQuery, we run circuit-level simulations in HSPICE with 45nm TSMC technology. We use VTEAM [57] model of memristors with $I_{ON}/I_{OFF}$ ratio of $10^3$ for non-volatile memory crossbar design. We develop software-based cycle-accurate simulator (based on C++) which emulates the functionality of the designed NVQuery. This allows us to speed up the simulation time significantly and verify the proposed design with diverse practical data sets. The simulator has accurate models of the hardware, e.g., time and power extracted from the circuit-level simulation to evaluate the efficiency of the proposed design. We compare NVQuery performance and energy efficiency with state-of-the-art query processing approaches running on the same technology node. We evaluate two popular approaches, sampling-based approximate querying (SAQ) [44] and deterministic approximate querying (DAQ) [7] on Intel i7 7600 CPU with 8GB memory. For measurement of the processor power, we use Hioki 3334 power meter. We use a dataset consisting a table of `Census` of 10 million tuples using 32-bit unsigned integers to compare the efficiency of different techniques. This data is popularly used to model populations of various types ranging from cities and organizations to word frequencies in natural language corpora. The SQL server contains a single table with one 10GB column of randomly generated records. In the rest of the paper, power and performance results have been reported for 1000 queries from aggregation and prediction functions over five randomly generated datasets. Join operation uses a different dataset, as the previously described dataset is not ideal for join based operations (need more than one column for join). Dataset includes 6 columned tables, randomly populated. Size of the table ranges from $2^2$ to $2^{17}$. The upper limit on the size is a function of the maximum datasheet size in MS Excel ($2^{20}$) and realistic join compute times.

### B. NVQuery Efficiency

Here we highlight the advantage that NVQuery can provide in computing each query function. Table V compares the energy savings and performance speedup of running different queries on proposed NVQuery as compared to a digital ASIC design. Each energy is reported when 10 queries run on 1k dataset. The selected dataset is small so that the reported values compare the computation energy without data movement cost. The digital system is designed using System Verilog in 45nm ASIC flow. The result shows that NVQuery improves the computation cost of all queries significantly. Specifically, queries such as `MAX, MIN` and/or `TOP k` can be processed in a single cycle, instead of processing in $O(n)$ or $O(logn)$ time. Our evaluation shows that our design can provide 11.8× energy improvement and 26.85× performance speedup on average compared to digital approach for nearest distance search-based queries. Similarly, our design can achieve on average

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



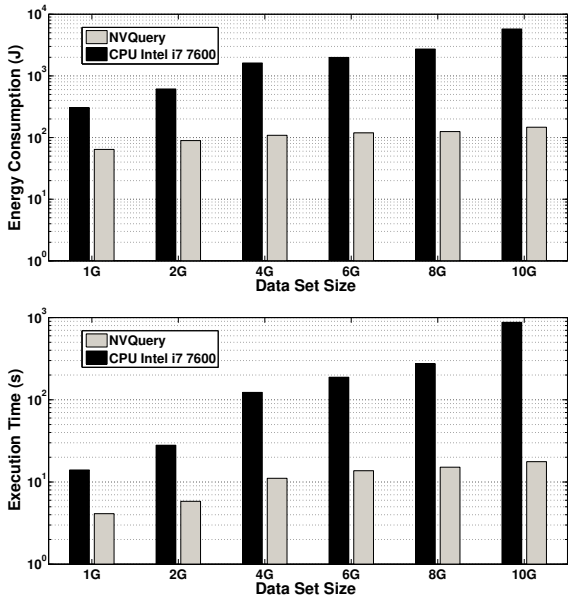


Fig. 6. Energy consumption and performance of query processing running on traditional core and the proposed NVQuery.

TABLE V
ENERGY CONSUMPTION AND PERFORMANCE SPEEDUP OF QUERIES IN NVQUERY NORMALIZED TO DIGITAL DESIGN OVER 1K DATA

| | Nearest search | | Search | Memory | |
|---|---|---|---|---|---|
| **Queries** | *MAX/ MIN* | *Top 1* | *Search/ Count* | *Addition/ Average* | *Bit-wise* |
| Energy Improv. | 9.5× | 14.1× | 13× | 5.8× | 46.7× |
| Speedup | 24.2× | 29.5× | 92.1× | 0.9× | 122.6× |

13.7× and 92.1× (5.8× and 0.9×) energy savings and performance speedup over exact search (memory functionalities, e.g. addition). NVQuery is also efficient in executing different join operations. For a small table with $2^2$ rows, NVQuery provides 2.3x speedup and 5.9x energy savings on average as compared to conventional systems. However, the performance and energy efficiency of NVQuery increases with table size. For example, for a table with $2^{15}$ rows, NVQuery provides speedup and energy efficiency improvement of 21x and 83x respectively. Although, the performance of in-memory addition is less than that of digital-based design, but considering the cost of data movement, it makes sense to process data locally in-memory. In large size query processing, the data movement dominates the computation cost, which motivates us to perform in-memory computations to avoid data movement issue.

### C. NVQuery & Dataset Size

While running real dataset, the main advantage of NVQuery comes from addressing the data movement issue. Fig. 6 shows the average energy consumption and performance of running query processing on traditional core and NVQuery when the data set size changes from 1GB to 10GB. Our evaluation shows that the NVQuery has an advantage in processing the nearest distance search and related functions such as `MIN`, `MAX` or `Top` queries. However, to see the average NVQuery improvement, we generate the same number of queries running on the dataset. Our evaluation shows that increasing the data size significantly increases the energy and execution time of traditional cores. However, this increment is minor in NVQuery as it can locally process the data. As our result in Table V shows, NVQuery not only avoids the overhead of data movement, but also provides much cheaper computation than traditional cores. This difference is more prominent when the size of the dataset passes 8GB, which is the available main memory size in our tested platform. In such case, the traditional cores require to bring data up from the hard disk, which significantly slows down the computation. Comparing the energy and performance of NVQuery for 10G data shows that, our design can achieve 34.7× energy savings and 49.3× performance speedup as compared to traditional processor running the same query tasks.

### D. NVQuery Approximation

Fig. 7 shows the energy, performance and energy-delay product, when NVQuery has been approximated using bit trimming and voltage scaling. The x-axis in the graph shows the number of relaxed bits. In addition, the red line in EDP graph shows the average relative error of query processing at different levels of approximation. Although the latency remains constant in the case of approximation by voltage scaling, it can achieve much higher efficiency than bit trimming. Our evaluation shows that NVQuery approximation using bit trimming and voltage scaling can provide 490.7× and 507.9× EDP improvement as compared to NVQuery in exact mode while ensuring less than 0.2% average relative error. The efficiency of the voltage scaling approximation becomes more significant in deep approximation. For instance, while accepting 2% error, approximation by voltage scaling can achieve 45.0× and 17.6× energy savings and speedup (807× EDP improvement).

We also compare the efficiency of the proposed NVQuery with the state-of-the-art approximate query accelerators SAQ [44] and DAQ [7] using 8G dataset size. The NVQuery and DAQ approximation is defined based on the number of blocks under voltage overscaling and the number of least significant bits neglected respectively. In SAQ the error rate is determined based on the requested error bound. Table VI shows the energy-delay product (EDP) improvement of the different query accelerators as compared to traditional CPU core when the level of approximation changes from 0% to 10%. For each error rate, we select those configurations of SAQ and DAQ which result in the best EDP improvement. As Table VI shows, increasing the number of relaxed bits improves the energy consumption of our design. Our experimental evaluation shows that, NVQuery can achieve 105.0× and 26.2× EDP improvement as compared to SAQ and DAQ designs in exact mode. The main advantage of NVQuery comes from addressing data movement issue. The NVQuery can provide higher efficiency when it works in approximate mode, since our memory-based design put a larger portion of memory under voltage overscaling in order to achieve the same error rate as DAQ design. In other words, when DAQ neglects *m*-bits for accelerating query processing, our memory-based design can get the same accuracy by putting larger

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2819080, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



TABLE VI
ENERGY-DELAY PRODUCT IMPROVEMENT OF SAQ, DAQ AND PROPOSED NVQUERY

| Query Accelerators | | 0% | 1% | 2% | 4% | 6% | 8% | 10% |
|---|---|---|---|---|---|---|---|---|
| SAQ [44] | Error bound | 0% | 1.5% | 3.1% | 5.2% | 7.4% | 8.5% | 10.9% |
| | EDP Improv. | 4.1× | 6.7× | 8.3× | 11.8× | 17.2× | 24.4× | 39.8× |
| DAQ [7] | trimmed bits | 0-bit | 1-bit | 2-bit | 4-bit | 6-bit | 7-bit | 9-bit |
| | EDP Improv. | 16.4× | 24.2× | 36.9× | 52.1× | 69.2× | 85.5× | 104.5× |
| NVQuery | Relaxed bits | 0-bit | 2-bit | 4-bit | 6-bit | 8-bit | 11-bit | 15-bit |
| | EDP Improv. | 431× | 505× | 807× | 1,515× | 2,288× | 2,587× | 3,154× |

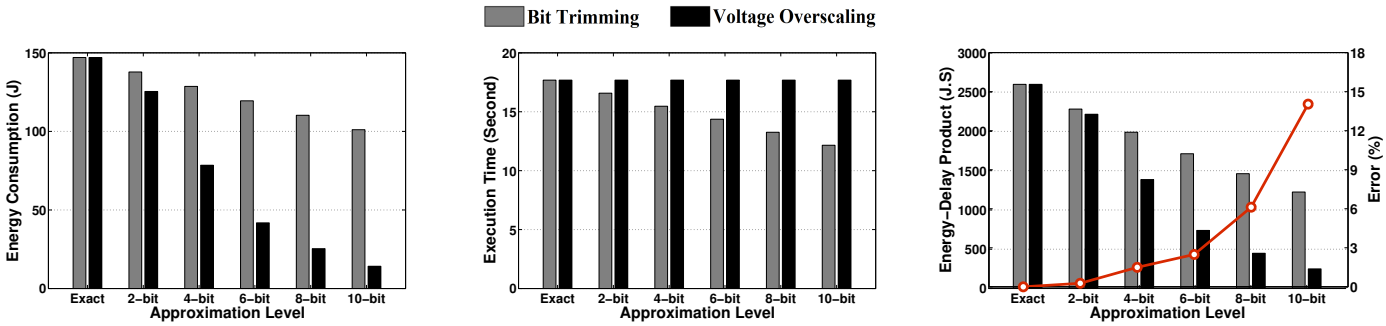


Fig. 7. Energy consumption and performance of the NVQuery at different approximation levels.

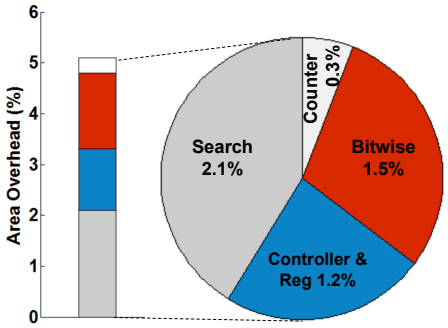


Fig. 8. Area overhead as compared to conventional crossbar memory

portion of memory blocks under voltage overscaling (shown in Table V for 4-bit stage size). Our evaluation shows that in approximate mode, NVQuery can achieve 79.2× and 30.1× EDP improvement as compared to SAQ and DAQ respectively, while providing similar error rate.

### E. Area Overhead

NVQuery has both memory and query processing functionalities. We added peripheral circuitry to crossbar memory to support nearest distance exact search operation, bit-wise/addition operations, counter and controller. Fig. 8 shows that proposed NVQuery has up to 5.1% area overhead compared to the conventional crossbar. The search circuitry takes 2.1% extra area. Counter and bit-wise circuits add 0.3% and 1.5% area overhead to design. Finally, the controller and registers take the rest 1.2% area overhead.

## VII. CONCLUSION

In this paper we propose a novel memory architecture which can accelerate query processing inside the memory. NVQuery supports a large range of query functionalities inside the memory. Our design exploits the analog characteristic of the non-volatile memory to design a configurable memory architecture which can look for exact or nearest distance values. Our result shows that NVQuery not only improves the cost of each query processing, but also completely addresses the data movement issue by locally processing the data in memory. To further improve the energy efficiency, our design NVQuery supports configurable approximation by adaptively putting memory under voltage overscaling. Our experimental evaluation shows that, in comparison with the state-of-the-art query accelerators, NVQuery in exact (approximate) mode can achieve 26.2× (30.1×) energy-delay product improvement while providing similar accuracy.

## VIII. ACKNOWLEDGMENT

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA and NSF grants #1730158 and #1527034, and Jacobs School of Engineering UCSD Powell Fellowship.

## REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[2] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.
[3] F. Imani, B. Yao, R. Chen, P. Rao, and H. Yang, "Factal pattern recognition of image profiles for manufacturing process monitoring and control," in *International Manufacturing Science and Engineering Conference*, p. 1, ASME, 2018.
[4] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact.," *MIS quarterly*, vol. 36, no. 4, 2012.
[5] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.

[6] B. Yao *et al.*, "Multifractal analysis of image profiles for the characterization and detection of defects in additive manufacturing," *Journal of Manufacturing Science and Engineering*, vol. 140, no. 3, p. 031014, 2018.

[7] N. Potti and J. M. Patel, "Daq: a new paradigm for approximate query processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 9, pp. 898–909, 2015.

[8] 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.

[9] S. Breß and G. Saake, "Why it is time for a hype: A hybrid query processing engine for efficient gpu coprocessing in dbms," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1398–1403, 2013.

[10] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "Looknn: Neural network with no multiplication," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1775–1780, IEEE, 2017.

[11] M. Imani, D. Peroni, and T. Rosing, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2017.

[12] M. Imani, Y. Kim, A. Rahimi, and T. Rosing, "Acam: Approximate computing based on adaptive associative memory with online learning.," in *ISLPED*, pp. 162–167, 2016.

[13] M. Imani, P. Mercati, and T. Rosing, "Remam: low energy resistive multi-stage associative memory for energy efficient computing," in *Quality Electronic Design (ISQED), 2016 17th International Symposium on*, pp. 101–106, IEEE, 2016.

[14] M. Imani, A. Rahimi, P. Mercati, and T. Rosing, "Multi-stage tunable approximate search in resistive associative memory," *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[15] C. Gregg and K. Hazelwood, "Where is the data? why you cannot debate cpu vs. gpu performance without the answer," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pp. 134–144, IEEE, 2011.

[16] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "Miso: souping up big data query processing with a multistore system," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1591–1602, ACM, 2014.

[17] X. Yin *et al.*, "Exploiting ferroelectric fets for low-power non-volatile logic-in-memory circuits," in *ICCAD*, pp. 1–8, IEEE, 2016.

[18] J. Sim *et al.*, "Lupis : Latch-up based ultra efficient processing in-memory system," in *International Symposium on Quality Electronic Design (ISQED)*, pp. 1–6, IEEE, 2018.

[19] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 14–26, IEEE Press, 2016.

[20] S. H. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "Comparing implementations of near-data computing with in-memory mapreduce workloads," *IEEE Micro*, vol. 34, no. 4, pp. 44–52, 2014.

[21] M. Imani, Y. Kim, and T. Rosing, "Nngine: Ultra-efficient nearest neighbor accelerator based on in-memory computing," in *International Conference on Rebooting Computing (ICRC)*, IEEE, 2016.

[22] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pp. 25–32, IEEE, 2017.

[23] M. Imani, S. Gupta, A. Arredondo, and T. Rosing, "Efficient query processing in crossbar memory," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*, pp. 1–6, IEEE, 2017.

[24] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[25] M. Imani, S. Gupta, and T. Rosing, "Genpim: Generalized processing in-memory to accelerate data intensive applications," in *Design Automation and Test in Europe Conference (DATE)*, pp. 1–6, IEEE/ACM, 2018.

[26] M. Saremi, S. Rajabi, H. J. Barnaby, and M. N. Kozicki, "The effects of process variation on the parametric model of the static impedance behavior of programmable metallization cell (pmc)," *MRS Online Proceedings Library Archive*, vol. 1692, 2014.

[27] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, "More efficient testing of metal-oxide memristor-based memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[28] S. Salehi and R. F. DeMara, "Process variation immune and energy aware sense amplifiers for resistive non-volatile memories," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pp. 1–4, IEEE, 2017.

[29] M. Saremi, "A physical-based simulation for the dynamic behavior of photodoping mechanism in chalcogenide materials used in the lateral programmable metallization cells," *Solid State Ionics*, vol. 290, pp. 1–5, 2016.

[30] B. Pourshirazi and Z. Zhu, "Refree: A refresh-free hybrid dram/pcm main memory system," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pp. 566–575, IEEE, 2016.

[31] M. K. Tavana, A. K. Ziabari, and D. Kaeli, "Live together or die alone: Block cooperation to extend lifetime of resistive memories," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1098–1103, IEEE, 2017.

[32] S. Salehi, N. Khoshavi, and R. F. Demara, "Mitigating process variability for non-volatile cache resilience and yield," *IEEE Transactions on Emerging Topics in Computing*, 2018.

[33] C. Liu, Q. Yang, C. Zhang, H. Jiang, Q. Wu, and H. H. Li, "A memristor-based neuromorphic engine with a current sensing scheme for artificial neural network applications," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pp. 647–652, IEEE, 2017.

[34] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pp. 757–763, IEEE, 2017.

[35] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[36] D. Schaa and D. Kaeli, "Exploring the multiple-gpu design space," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–12, IEEE, 2009.

[37] B. Sukhwani, M. Thoennes, H. Min, P. Dube, B. Brezzo, S. Asaad, and D. Dillenberger, "A hardware/software approach for database query acceleration with fpgas," *International Journal of Parallel Programming*, vol. 43, no. 6, pp. 1129–1159, 2015.

[38] C. Dennl, D. Ziener, and J. Teich, "Acceleration of sql restrictions and aggregations through fpga-based dynamic partial reconfiguration," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pp. 25–28, IEEE, 2013.

[39] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, "Storegpu: exploiting graphics processing units to accelerate distributed storage systems," in *Proceedings of the 17th international symposium on High performance distributed computing*, pp. 165–174, ACM, 2008.

[40] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W.-m. W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 347–358, ACM, 2010.

[41] J. Tigani and S. Naidu, *Google BigQuery Analytics*. John Wiley & Sons, 2014.

[42] Y. Sun, Y. Wang, and H. Yang, "Energy-efficient sql query exploiting rram-based process-in-memory structure," in *Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2017 IEEE 6th*, pp. 1–6, IEEE, 2017.

[43] S. Acharya, P. B. Gibbons, and V. Poosala, "Aqua: A fast decision support systems using approximate query answers," in *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 754–757, Morgan Kaufmann Publishers Inc., 1999.

[44] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "Blinkdb: queries with bounded errors and bounded response times on very large data," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29–42, ACM, 2013.

[45] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra, "Worst-case optimal join algorithms:[extended abstract]," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pp. 37–48, ACM, 2012.

[46] T. L. Veldhuizen, "Leapfrog triejoin: A simple, worst-case optimal join algorithm," 2012.

[47] N. Bruno, Y. Kwon, and M.-C. Wu, "Advanced join strategies for large-scale distributed computation," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1484–1495, 2014.

[48] E. J. Shekita, H. C. Young, and K.-L. Tan, "Multi-join optimization for symmetric multiprocessors," in *VLDB*, vol. 93, pp. 479–492, 1993.

[49] F. N. Afrati and J. D. Ullman, "Optimizing joins in a map-reduce environment," in *Proceedings of the 13th International Conference on Extending Database Technology*, pp. 99–110, ACM, 2010.

[50] M. Zhou, R. Zhang, D. Zeng, W. Qian, and A. Zhou, "Join optimization in the mapreduce environment for column-wise data store," in *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pp. 97–104, IEEE, 2010.
[51] X. Yin *et al.*, "Design and benchmarking of ferroelectric fet based tcam," in *DATE*, pp. 1444–1449, IEEE, 2017.
[52] M. Imani, D. Peroni, A. Rahimi, and T. Rosing, "Resistive cam acceleration for tunable approximate computing," *IEEE Transactions on Emerging Topics in Computing*, 2016.
[53] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.
[54] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 6, ACM, 2017.
[55] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 409–414, IEEE Press, 2011.
[56] M. Imani, S. Patil, and T. S. Rosing, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pp. 373–378, EDA Consortium, 2016.
[57] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

**Mohsen Imani** received his M.S. and BCs degrees from the School of Electrical and Computer Engineering at the University of Tehran in March 2014 and September 2011 respectively. From September 2014, he is a Ph.D. student in the Department of Computer Science and Engineering at the University of California San Diego, CA, USA. He is a project leader at System Energy Efficient Laboratory (SeeLab) where he is mentoring several graduate and undergraduate students on different computer engineering projects from circuit to system level. Mr. Imani research focuses on computer architecture, machine learning and brain-inspired computing.

**Saransh Gupta** is pursuing a Masters degree in Electrical and Computer Engineering from University of California, San Diego. He is a member of System Energy Efficiency Laboratory (SEELab), where he is working on alternate computing paradigms. He received his B.E. in Electrical and Electronics Engineering from Birla Institute of Technology Science, Pilani - K.K. Birla Goa Campus in 2016. His research interests include application of non-volatile memories, computer architecture, and electronic circuits with an emphasis on processing in-memory.

**Sahil Sharma** received his MS in Computer Science from University of California at San Diego in 2017, and B.Tech in Computer Science and Engineering from Indian Institute of Technology, Kharagpur in 2014. His research interests include computer architecture, operating and embedded systems. He was a member of the System Energy Efficient Laboratory (SEELAB), University of California at San Diego.

**Tajana Simunic Rosing** is a Professor, a holder of the Fratamico Endowed Chair, and a director of System Energy Efficiency Lab at UCSD. She is currently heading the effort in SmartCities as a part of DARPA and industry funded TerraSwarm center. During 2009-2012 she led the energy efficient datacenters theme as a part of the MuSyC center. Her research interests are energy efficient computing, embedded and distributed systems. Prior to this she was a full time researcher at HP Labs while being leading research part-time at Stanford University. She finished her PhD in 2001 at Stanford University, concurrently with finishing her Masters in En-gineering Management. Her PhD topic was Dynamic Management of Power Consumption. Prior to pursuing the PhD, she worked as a Senior Design Engineer at Altera Corporation.