DigitalPIM: Digital-based Processing In-Memory for Big Data Acceleration

Mohsen Imani, Saransh Gupta, Yeseong Kim, Minxuan Zhou, Tajana Rosing
CSE Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, sgupta, yek048, miz087, tajana}@ucsd.edu

ABSTRACT

In this work, we design, DigitalPIM, a Digital-based Processing In-Memory platform capable of accelerating fundamental big data algorithms in real time with orders of magnitude more energy efficient operation. Unlike the existing near-data processing approach such as HMC 2.0, which utilizes additional low-power processing cores next to memory blocks, the proposed platform implements the entire algorithm directly in memory blocks without using extra processing units. In our platform, each memory block supports the essential operations including: bitwise operation, addition/multiplication, and search operation internally in memory without reading any values out of the block. This significantly mitigates the processing costs of the new architecture, while providing high scalability and parallelism for performing the extensive computations. We exploit these essential operations to accelerate popular big data applications entirely in memory such as machine learning algorithms, query processing, and graph processing. Our evaluations show that for all tested applications, the performance can be accelerated significantly by eliminating the memory access bottleneck.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Hardware** → **Emerging technologies**; *Electronic design automation*;

KEYWORDS

Processing in Memory; Non-volatile memories; Energy efficiency; Big data acceleration

ACM Reference Format:
Mohsen Imani, Saransh Gupta, Yeseong Kim, Minxuan Zhou, Tajana Rosing. 2019. DigitalPIM: Digital-based Processing In-Memory for Big Data Acceleration. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3319483

1 INTRODUCTION

We live in a world where technological advances are continually creating more data than what we can cope with. With the emergence of the Internet of Things, sensory and embedded devices will generate massive data streams demanding services that pose huge technical challenges due to limited device resources. Today IoT applications typically analyze raw data by running machine learning algorithms in data centers. Sending all the data to the cloud for processing is not scalable, cannot guarantee a real-time response, and is often not desirable due to privacy and security concerns. Much of IoT data processing will need to run at least partly on the devices at the edge of the internet. However, running data-intensive workloads with large datasets on traditional cores results in high energy consumption and slow processing speed due to a large amount of data movement between memory and processing units. Although new processor technology has evolved to serve computationally complex tasks in a more efficient way, data movement costs between processor and memory still hinder the higher efficiency of application performance. In addition, applications in this area involve diverse data analytic procedures which needs to be significantly accelerated while handling a large amount of data. Our work seeks to build systems capable of responding to the diverse needs in real time with orders of magnitude more energy efficient operation.

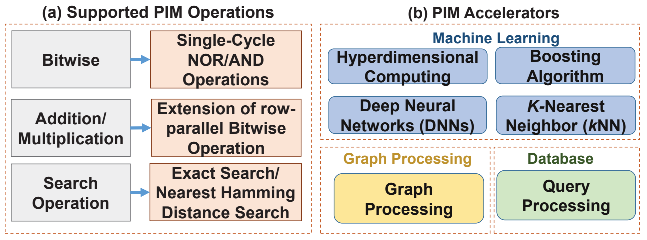
(a) Supported PIM Operations
Bitwise
Single-Cycle NOR/AND Operations
Addition/ Multiplication
Extension of row-parallel Bitwise Operation
Search Operation
Exact Search/ Nearest Hamming Distance Search
(b) PIM Accelerators
Machine Learning
Hyperdimensional Computing
Boosting Algorithm
Deep Neural Networks (DNNs)
K-Nearest Neighbor (kNN)
Graph Processing
Graph Processing
Database
Query Processing

Figure 1: The list of supported operations and (b) the list of applications accelerated by the proposed DigitalPIM platform.

In this work, we propose, DigitalPIM, a novel Digital-based Processing In-Memory (PIM) platform which accelerates the fundamental operations and diverse data analytic procedures. The DigitalPIM supports fundamental block-parallel operations inside memory, e.g., addition, multiplication or bitwise computations. This capability is implemented on a crossbar memory which stored data point of an application. Since a large amount of data is not required to be sent to the processing cores for computation, the performance can be accelerated significantly by avoiding the memory access bottleneck. To fully get the advantage of DigitalPIM for popular data processing procedures and machine learning algorithms, we design specialized accelerator. Our platform can process several applications including machine learning, graph and query processing entirely in-memory without using any processing cores.

2 DIGITAL-BASED PIM

2.1 DigitalPIM Overview

In this work, we present processing in-memory architecture to enable hardware accelerations for popular big data applications. We exploit a conventional crossbar memory to enable essential operations in memory. Figure 1a shows the list of operations supported by DigitalPIM platform. The first supported operation by DigitalPIM is bitwise operation. In contrast to prior works that compute bitwise operation on the sense amplifier of each memory block [1], our design supports bitwise operations internally in memory without even reading the values out of the block. Prior work enabled the bitwise operation by exploiting the analog/resistive characteristic of

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3319483

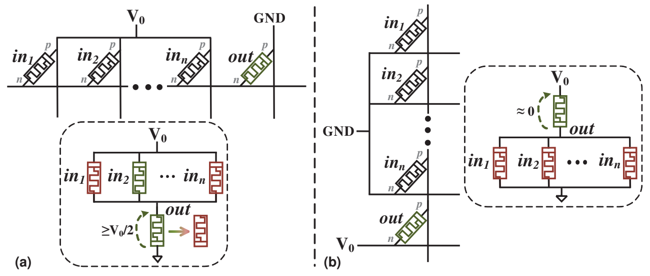

Figure 2: *n*-input NOR implementation in (a) a row and (b) a column [3].

memristor devices, which causes the internal device switching [2, 3]. Our platform extends the bitwise operation, e.g., `NOR`, to implement addition and multiplication in memory. These operations are enabled by performing a series of bitwise operations. Although DigitalPIM addition and multiplication are much slower than CMOS logics, the high parallelism of the PIM operations results in a higher overall DigitalPIM throughput as compared to the CMOS-based logics. In other words, DigitalPIM performs the computation in a row-parallel way, resulting in about 1K parallel addition/multiplication on memory with 1K rows. DigitalPIM also supports search operation internally in memory. During the search, blocks configured as content addressable memory (CAM). Conventionally CAM blocks are only used for exact matching. Here, we extend the search operation such that CAMs can also find a row with the nearest Hamming distance or the nearest absolute distance with query data.

DigitalPIM exploits the PIM-supported operations to accelerates diverse data-intensive applications partially or entirely in-memory. Figure 1b shows the list of applications accelerated by our proposed DigitalPIM architecture. Our platform can accelerate different machine learning including deep neural networks, AdaBoost, k-nearest neighbor applications entirely in memory without using any processing cores. DigitalPIM can also accelerate graph and query processing algorithms, along with a set of Hyperdimensional (HD) computing algorithms that support classification on a wide range of data types.

2.2 DigitalPIM Supported Operations

Bitwise Operations: We exploit the analog characteristic of memristor devices to support bit-parallel or row-parallel bitwise operation internally in memory. Work in [2, 4] showed that crossbar memory can internally support NOR-based operation between the N selected rows or columns of the memory, as shown in Figure 2. This operation happens by activating N columns (rows) of the memory at the same time and connecting the target column (row) which store the results of NOR operation to a zero voltage. This enables the internal NOR operation between the selected columns (rows) to be written directly on the selected output column (row).

Our work, FELIX [3], exploits the bipolar nature of the voltage controlled ReRAM devices to enable `NOR`, `NOT`, `Min`, `NAND`, and `OR` in a single cycle in crossbar memory. The benefits from FELIX are muti-fold. Not only it is 1.86× faster and 2.21× more energy efficient but also requires 1.68× lower memory as compared to the fastest digital PIM technique. The addition and multiplication are supported in crossbar memory by performing a series of sequential FELIX operations. In FELIX, multiple NOR/NAND gates are combined together to implement any logic operation, including but not limited to XOR, majority, and implication. For example, AND operation can be implemented with two sequential NAND operations. In a different approach to PIM, researchers have tried to exploit sense amplifiers to read out data locally and process it. Our works in MPIM [1] and LUPIS [5] modified memory sense amplifiers to enable bitwise operations like AND, OR, XOR, etc. The sense amplifier based techniques trade energy and area for better performance as compared to a relatively slower PIM execution.

Addition/Multiplication: The basic logic functions supported by MAGIC and FELIX can be extended to implement addition and multiplication. An implementation of addition proposed by FELIX combines XOR and majority operations [3]. A 1-bit adder can be represented by,

$$S = A \oplus B \oplus C_{in}, \tag{1a}$$

$$C_{out} = A.B + B.C + C.A = MajNA, B, C_{in}, \tag{1b}$$

where A, B, and C_{in} are 1-bit inputs while S and C_{out} are the generated sum and carry bits respectively. Here, S is implemented as two serial in-memory XOR operations. C_{out}, on the other hand, can be executed by inverting the output of . Hence, S takes a total of 4 cycles and 2 additional memristors, while C_{out} needs 2 cycles and 2 additional memristors. Further, multiplication is implemented by multiple additions and shift operations. A naive implementation of multiplication would result in a lot of data movement due to shifts. Our work in [6] proposed the first digital PIM-based multiplication. It dealt with the data movement overhead by sub-dividing memory blocks using interconnects which allowed bit-shifts without reading out data and also accelerated PIM multiplication. The work in [7] takes another approach where it does not accelerate an individual multiplication significantly but makes it highly parallelizable. Work in [8] exploited DigitalPIM operations to support floating point operation in memory and extended them to accelerate deep neural network training.

Search: Our work, MPIM [1], supports a block-serial row parallel Hamming distance based operations. We proposed ReCAM [9], which can be configured to perform the closest distance search operation inside the memory. Although it does not implement binary distance search but it is a close approximation. However, many applications require a binary distance based comparison. We proposed NVQuery [10, 11] which enabled, for the first time, exact binary distance search in memory. It achieved it by using smart voltage application techniques to exploit the differential discharging time of capacitance. We used it to enable complex functions, including but not limited to sorting, compare, aggregation, and prediction. These functions have huge time complexities when implemented on traditional hardware.

3 MACHINE LEARNING ACCELERATORS

In this section, we describe how we fully leverage the advantages of DigitalPIM for large scale data processing and machine learning applications by designing specialized PIM accelerator blocks. In machine learning, we look at four important classes of algorithms: hyperdimensional computing, deep learning, AdaBoost, and k-Nearest Neighbor algorithms.

3.1 Hyperdimensional Computing Acceleration

The brain's circuits are massive in terms of numbers of neurons and synapses, suggesting that large circuits are fundamental to the brain's computing. Brain-inspired hyperdimensional (HD) computing explores this idea by looking at computing with ultra-wide words high-dimensional vectors, or hypervectors [12–14]. Figure 3a shows the overview structure of HD computing for classification example. The first step is to encode data into hypervectors. The goal of the encoder is to map key input data to a single hypervector and then combine these for all of the data points in a class to generate a unique hypervector representing each class. Each class hypervector is a long

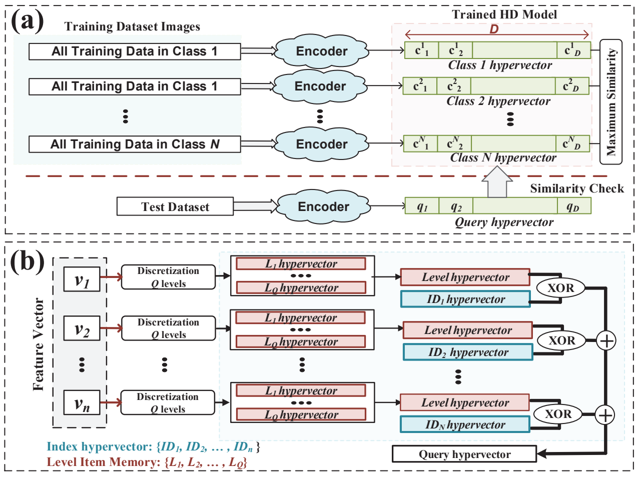

Figure 3: (a) overview of HD computing during classification. (b) The details of the encoding functionality in HD computing [3].

vector with D dimension with binary (0, 1) elements. Associative memory stores the trained hypervectors for all classes. Once the data is encoded and clustered into appropriate categories, various types of operations can be implemented, but at the heart of all implementations is a search for a category match. HD classifies an unknown input data by encoding it to a hypervector using the same encoder used for training. The query hypervector has binary elements and the same dimensions as the class hypervectors. Then, this hypervector is broadcast to the associative memory module for comparing with a set of learned class hypervectors. The associative memory returns the closest match. We exploit such architectural insights to develop scalable and efficient encoder and associative memories that can accelerate HD computing as discussed next.

Encoder: Figure 3b shows the functionality of the encoding module in HD computing. HD encoder generates m random hypervectors representing each feature level ($L_1, L_2, \cdots, L_m$). These hypervectors use only binary elements, with information randomly distributed over all D dimensions (e.g. $D = 10,000$). To consider the impact of feature position, HD computing assigns a unique position hypervectors. For a feature vector with n elements, HD generates n P position hypervectors. The P is another randomly generated hypervector with the same dimensionality as the level hypervector, D, and specifies the effect of feature position in the final hypervector. In this way, the hypervectors are combined together using element-wise XOR of the position and level hypervectors, and then summing the resulting hypervectors over all features. We map all these operations in PIM-enabled memory [3]. We perform n PIM XORs (one for each ID) and generate n outputs. For the first n iterations, we select one ID and XOR it with one of the Ls in every iteration. For a pair of ID and L, PIM XOR can be computed in parallel for all dimensions. We then add the generated n elements serially, three bits at a time. If $X_1, X_2, \ldots X_n$ are the vectors to be added together, we first add X_1, X_2, and X_3 to generate S_1 and C_1. We then add X_4, X_5, and $\{C_1, S_1\}$ to generate S_2 and C_2, and so on till we have added all XOR results. The addition of n 1-bit elements results in an output with $p = \lceil log_2 n \rceil$ bits. We implement this PIM-based HD encoding while ensuring optimal storage, whose details can be found in [3].

Associative memory: is in charge of storing all class hypervectors. Our design stores all available classes in an associative memory as a trained HD model. In inference, when an unknown input digit is loaded into the system, our design uses a similar encoding to generate the *query hypervector*. Then, associative memory finds the closest match between a set of learned hypervectors and a query hypervector by using a distance metric. Depending on the data representation, HD can use different similarity metrics. Here we consider Hamming distance to find the most similar class. The state-of-the-art associative memories are not able to compare vectors with dimensionality in thousands [9, 15]. They also lack techniques for uniformly tolerating errors in any vector component and hence cannot efficiently exploit the holographic feature of hypervectors. Further, most of them are implemented as a content addressable memory (CAM) and so are not able to find the minimum distance. We designed new architectures for hyperdimensional associative memory that can facilitate energy-efficient, fast, and scalable search operations [13]. Our proposed CAM exploits the analog discharging current of different CAM rows to identify a CAM row with the minimum mismatches with the query hypervector. In other words, our design uses a tree-based structure of a Loser-Takes-All (LTA) blocks to identify a CAM row (a class) with the minimum Hamming distance. These analog HD designs linearly scale with the number of dimensions in the hypervectors while exploring a large design space at orders of magnitude higher efficiency [13].

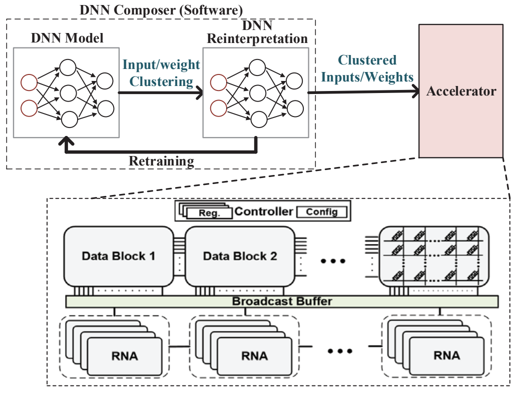

Figure 4: The overview of the proposed DNN acceleration framework [16].

3.2 Deep Neural Network Acceleration:

We present a novel PIM accelerator for DNNs that supports all DNN functionality in memory [16]. Figure 4 shows an overview of the proposed framework. Our framework first analyzes the computation flow of a DNN model and encodes key DNN operations for a specialized PIM-enabled accelerator. It identifies the representative parameters, i.e., weights and input values, for each neuron using clustering algorithms. The other key operations, like activation functions, are also modeled to enable in-memory processing. The key idea underlying our design is that even though the operations of a DNN are continuous, they can be approximated as step-wise functions without losing the quality of inference. Thus, we can create lookup tables that store the finite pre-computed values, and map them into specialized memory blocks capable of computation.

Prior work proposed PIM-based neural network accelerators which keep the input data and trained weights inside memory [17, 18]. For example, the work in [18] showed that memristor devices can model multiplications and additions for each neuron. Authors of [18] store trained weights of each neuron as device resistance values, and pass the current converted from digital values in a similar way to spiking neuromorphic computing [19]. Although this approach is a first step toward using PIM for DNN acceleration, it has two major issues: (i) it only supports two in-memory functions while other important operations, such as activation functions, are implemented using CMOS-based logics, which would make the fabrication expensive. (ii) Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) do not scale as memory device technology does, although they take 61% of power consumption [18]; therefore, the analog-based computation approach would not be an appropriate solution to design PIM-based DNN accelerators.

Our proposed design [16] addresses the key issues present in prior work. Our framework has two components: a software DNN composer, and a hardware accelerator (Shown in Figure 4). The role of the DNN composer is to convert each neural network operation into tables which can be stored in the accelerator memory blocks for processing in memory. In an offline stage, starting with a given DNN model, the DNN composer analyzes weights and inputs of each neuron and generates a new DNN model which is compatible with the proposed PIM-based accelerator. The newly constructed DNN model is repeatedly revised through multiple retraining procedures. The final model is stored into the accelerator for online inference. Our framework supports three layers of popularly used for designing DNNs: fully-connected, convolution, and pooling layers. We group the computation tasks of the networks into four operations: multiplication, addition, activation function, and pooling.

Considering the model of a single neuron, our approach works based on the encoded input/weight [16], since the DNN composer selects the best representatives inputs/weights by analyzing the network. Therefore, the multiplication can be implemented by simply accessing the pre-evaluated multiplication values. Then, the results of multiplication can locally accumulate in memory without using any processing cores. Our design approximately models the activation function using a lookup table. Finally, we utilize another lookup table with a similar structure to encode the output of the activation function. We exploit the fact that the weights of a network are computed during testing. These constant weights can be preprocessed and encoded in a way suitable for in-memory processing. Since pre-processing is done only once, its latency is negligible in comparison to the speedup provided during the testing phase. Our evaluation shows that our approach [16] achieves 68.4×, 49.5× energy efficiency improvement and 48.1×, 10.9× speedup as compared to ISAAC [18] and PipeLayer [20], the state-of-the-art DNN accelerators, while ensuring less than 0.5% of quality loss.

3.3 Adaptive Boosting Acceleration

One of the methods for classification is to exploit decision rules which are automatically generated by learning algorithms such as ID3, C4.5, and CART [21]. The simple decision rules and trees can be combined to create more accurate and sophisticated decisions with meta-learning algorithms. AdaBoost (Adaptive Boosting) is one of the best meta-learning algorithms. In our previous work, we designed a novel hardware accelerator, which accelerates the decision rule computations inside memory for visual object recognition tasks [22]. The proposed design accelerates both the image feature extraction and boosting-based learning algorithm, which are key subtasks of the state-of-the-art image recognition approaches. We show in [22] that our design successfully performs practical image recognition tasks, including text, face, pedestrian, and vehicle recognition with 0.3% of accuracy loss due to approximation.

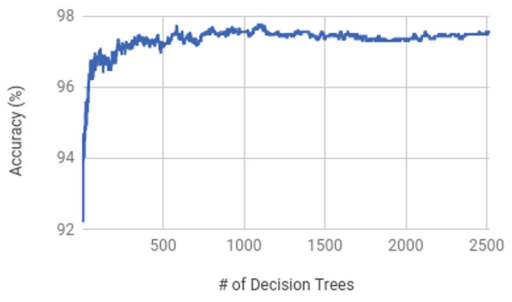

Figure 5: AdaBoost classification accuracy.

Unlike the previous work that relies on the existence of learning models trained offline, as a part of this work we design a new training algorithm for general classification problems and accelerate it by leveraging PIM. The key idea is that the training procedure of the decision trees can be decomposed into two steps: i) generating multiple candidates of decision rules, and ii) evaluating the expected accuracy of each decision rule to find the optimal rule. Evaluating a decision rule can be implemented using the similarity search operations that the PIM technique supports. Thus, by mapping multiple memory blocks for each decision candidate, we can perform the evaluation procedure in a block-parallel way, thus quickly identifying the best rule. To verify that our classification method is able to handle general classification problems, we implemented our idea in software and conducted an initial analysis by classifying human activities from smartphone-based sensors. As shown in Figure 5, the modified AdaBoost training method can create the first decision tree with 92% accuracy, and by combining multiple decision rules (on x-axis), we can further increase the accuracy by 98% for human activity classification.

3.4 K-Nearest Neighbor Acceleration

Nearest neighbor search problem arises in numerous fields of applications, including pattern recognition, statistical classification, biology, computer vision, etc. The nearest search computation is highly parallelizable for datasets on the order of tens of megabytes. Running analysis of more massive datasets on existing general purpose processors results in significant performance overhead due to a large number of data movements across the memory hierarchy. For example, to process 1 billion candidate points, one query needs 150 GFLOPs computation and 500G of data communication [23]. We propose to design a K-Nearest Neighbor accelerator [24] consisting of ternary content addressable memory (TCAM) blocks which enable in-memory nearest search. Our design exploits dynamic voltage scaling and analog detector circuitry to find kNN data based on the Hamming distance. It overcomes energy and performance issues in traditional computing systems by utilizing multiple TCAMs to search for the nearest neighbor data in parallel. We also design a new search-based accelerator which supports the nearest search with different distance metrics. The proposed memory is a configurable architecture which could take any similarity metric.

4 GRAPH PROCESSING & DATABASE ACCELERATOR

We have also used PIM to accelerate graph and query processing algorithms completely in-memory without using any processing cores. In the following, we explain the details of the different design implementation.

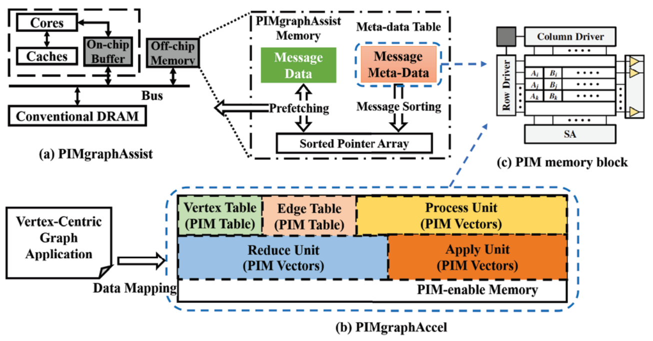

Figure 6: The architecture overview of PIMgraphAssist and PIMgraphAccel. [30]

4.1 Graph Processing Acceleration

Large-scale graph processing is becoming more and more important in the big data era. Framework-based models are widely used to represent most graph processing applications which run on different computing platforms. However, such large-scale graph applications are not handled efficiently by conventional architecture because of the irregular structure of graph data. For instance, widely used message passing vertex-centric model [25] has random memory access patterns because the destination of each message depends on the structure of the graph which is very irregular. Several accelerators have been proposed to improve the memory performance of hardware when processing graph applications. For instance, Tesseract [26] utilizes near-data computing based on 3D DRAM to build a scalable architecture for efficiently processing message-passing based graph applications. GraphPIM [27] offloads synchronization operations to memory to reduce the overhead of atomic operations. However, the inefficient data movement still exists in all these works because most operations are handled by processing cores. There are also some customized accelerators for graph processing, like Graphicionado [28] and work done by Ozdal et al. [29]. These ASIC designs reduce the impact of random memory accesses by special on-chip memory architectures which adds a lot of hardware complexity and does not scale well with larger graph data sizes. In contrast, we plan to utilize novel PIM technologies to minimize the inefficiency caused by the conventional memory hierarchy in graph processing. Specifically, we propose two different accelerators for graph processing: a PIM-based accelerator assistant for vertex-centric graph processing, and a general graph processing accelerator which translates the whole graph application into in-memory operations with high-degree parallelism.

PIMgraphAssist: Our proposed accelerator assistant is a heterogeneous memory component added in the conventional memory hierarchy. PIMgraphAssist consists of an on-chip buffer and an off-chip PIM memory as shown in Figure 6. The off-chip PIM memory has an associative memory and a normal memory. The associative memory supports fast in-memory search operations and stores meta-data of generated messages of application, while the normal memory stores contents of messages. For each message processed in the application, we write the message into the off-chip memory without accessing conventional memory hierarchy (including on-chip caches). The meta-data table stores an entry for each message, and each entry consists of source vertex, the destination vertex, and the address of message in the normal memory. Then, the accelerator sorts all the messages based on the order that the program requires by utilizing efficient associative searches in the meta-data table. The sorting process generates a pointer array and a two-level prefetching scheme is designed to efficiently send sorted results to the on-chip buffer. Specifically, PIMgraphAssist uses a streaming prefetching when accessing the pointer array and an indirect prefetching for each pointer fetched. The performance gain of our proposed PIM accelerator is due to two reasons: the random memory access will no longer cause high cache miss rates because all corresponding memory accesses are directly forwarded to PIM-based accelerator, and the processor can prefetch the sorted messages which maximize the memory bandwidth. Our design can be used for existing message passing based graph processing software frameworks by introducing memory load and store API for all messages generated.

PIMgraphAccel: Our PIM-based accelerator maps each phase of graph processing to in-memory operations by leveraging a large amount of processing memory blocks that can compute simultaneously. A typical vertex-centric graph processing framework has three phases: process, reduce and apply. In the process phase, all active vertices process messages based on their current application-specific status data and edge information. Then, all vertices receiving messages collect the message received and reduce them into a new temporary value based on an application-specific function during the reduce phase. In the apply phase, each vertex updates its status data based on the computed temporary values. In such a framework, there is a great potential of parallelism which cannot be fully exploited by conventional architecture because of data movement and an insufficient number of processing cores. For example, all operations during the process phase can be handled simultaneously. Furthermore, reduce and apply operations on different vertices can also be parallelized because there is no data dependency. Based on this observation, we design a graph processing accelerator PIMgraphAccel, by leveraging PIM architecture supporting various operations. Each function unit has a heterogeneous PIM architecture handling different tasks by PIM blocks. Specifically, we store the application-specific property (e.g., distance value in single-source-shortest-path) in PIM blocks supporting normal PIM operations like addition and bit-wise operations. These PIM blocks can compute several properties stored in the same row by one PIM command. In addition to row-level parallelism, different PIM blocks can perform computations simultaneously to provide block-level parallelism. We utilize PIM blocks to enable fast associative search to accelerate meta-data look-up in graph processing, like fetching active vertices list. Furthermore, we design several techniques to maximize the parallelism of PIM operations. PIMgraphAccel utilizes in-memory associative searches to schedule computations which can be parallelized during run-time. A new circuit design is proposed to enable parallel in-memory compare-and-swap (CAS) operations. We expect to significantly improve the performance of framework-based graph processing application by utilizing extensive parallelism and efficient in-memory data movements. To extend the lifetime of the proposed accelerator, we also design an endurance management mechanism specific to graph processing to evenly distribute write operations to different NVM parts.

4.2 Query Processing Acceleration

Data management systems (DMS) are used for collecting and analyzing large amounts of data for web applications and end users. The execution time of DMS queries increases at least linearly and sometimes exponentially as more records are stored on a server due to hardware and software limitations. Our recent work proposed a non-volatile memory-based query processing accelerator, called NVQuery [10], that supports a wide range of query functionalities including aggregation functions, prediction functions, bitwise operations, addition, exact, and nearest distance search operations. Our experimental evaluation using common SQL queries shows that, as compared to the state-of-the-art query accelerators [31], NVQuery can achieve 26.2× energy-delay product improvement while providing similar accuracy. However, it does not support some commonly used query functions.

We also proposed NVQuery+ [11] that support different types of functions which are costly to process on conventional core including: Between and Join which are used frequently in a query system. These functions are very costly to compute on conventional systems. On the other hand, NVQuery+ exploits its in-memory architecture to accelerate three different types of Joins, namely inner, right, and left, efficiently. We enable these extra functionalities by changing the memory structure to support functionalities such as sort and in-memory addition. Furthermore, our design support two types of parallelization in memory: row level parallelism, which allows multiple additions performs in a single memory block and block-level parallelism which increases the overall throughput of PIM. In addition, we extend the idea of NVQuery+ to approximate query processor by applying voltage overscaling on the memory block and enabling approximate in-memory multiplication. In this mode, the query operations can be implemented in an approximate mode within a controllable level of accuracy. This design adaptively balances the level of energy efficiency and quality of computation based on the accuracy requirement.

5 CONCLUSION

Running data-intensive workloads with large datasets on traditional cores results in high energy consumption and slow processing speed due to a large amount of data movement between memory and processing units. In this work, we built a platform capable of responding to our needs in real time with orders of magnitude more energy efficient operation. We propose a Processing In-Memory platform which accelerates fundamental operations and diverse data analytic procedures. Instead of sending a large amount of data to the processing cores for computation, our platform performs a large part of computation tasks inside the memory, thus the application performance can be accelerated significantly by avoiding the memory access bottleneck. Our platform accelerates general real-world big applications including machine learning applications, graph processing, and query processing.

ACKNOWLEDGMENT

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

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

[2] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MagicâĂŤmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[3] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2018.

[4] 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.

[5] J. Sim, M. Imani, W. Choi, Y. Kim, and T. Rosing, "Lupis: latch-up based ultra efficient processing in-memory system," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pp. 55–60, IEEE, 2018.

[6] 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.

[7] A. Haj-Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky, "Imaging-in-memory algorithms for image processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2018.

[8] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, ACM, 2019.

[9] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1327–1332, IEEE, 2016.

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

[11] M. Imani, S. Gupta, S. Sharma, and T. Rosing, "Nvquery: Efficient query processing in non-volatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[12] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

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

[14] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[15] T. Kohonen, *Associative memory: A system-theoretical approach*, vol. 17. Springer Science & Business Media, 2012.

[16] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.

[17] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[18] 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," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[19] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[20] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, IEEE, 2017.

[21] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

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

[23] Y. Deng and B. Manjunath, "Content-based search of video using color, texture, and motion," in *Proceedings of International Conference on Image Processing*, vol. 2, pp. 534–537, IEEE, 1997.

[24] M. Imani, Y. Kim, and T. Rosing, "Nngine: Ultra-efficient nearest neighbor accelerator based on in-memory computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, IEEE, 2017.

[25] G. Malewicz *et al.*, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 135–146, ACM, 2010.

[26] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[27] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *2017 IEEE International symposium on high performance computer architecture (HPCA)*, pp. 457–468, IEEE, 2017.

[28] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13, IEEE, 2016.

[29] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy efficient architecture for graph analytics accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 166–177, IEEE, 2016.

[30] M. Zhou, M. Imani, S. Gupta, and T. Rosing, "Gas: A heterogeneous memory architecture for graph processing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, p. 27, ACM, 2018.

[31] 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.