Accelerating Similarity-based Mining Tasks on High-dimensional Data by Processing-in-memory

Fang Wang*, Man Lung Yiu*, Zili Shao†
*Department of Computing, Hong Kong Polytechnic University
†Department of Computer Science and Engineering, the Chinese University of Hong Kong
*{csfwang, csmlyiu}@comp.polyu.edu.hk †shao@cse.cuhk.edu.hk

***Abstract*—Similarity computation is a core subroutine of many mining tasks on multi-dimensional data, which are often massive datasets at high dimensionality. In these mining tasks, the performance bottleneck is caused by the *'memory wall'* problem as substantial amount of data needs to be transferred from memory to processors. Recent advances in non-volatile memory (NVM) enable processing-in-memory (PIM), which reduces data transfer and thus alleviates the performance bottleneck. Nevertheless, NVM PIM supports specific operations only (e.g., dot-product on non-negative integer vectors) but not arbitrary operations. In this paper, we tackle the above challenge and carefully exploit NVM PIM to accelerate similarity-based mining tasks on multi-dimensional data without compromising the accuracy of results. Experimental results on real datasets show that our proposed method achieves up to 10.5x and 8.5x speedup for state-of-art kNN classification and k-means clustering algorithms, respectively.**

I. Introduction

Similarity computation is an essential building block for many mining tasks on multi-dimensional data. Examples of similarity-based mining tasks include kNN classification, k-means clustering, motif discovery and anomaly detection. These mining tasks often deal with massive datasets at high dimensionality. In addition, we encounter the *memory wall* issue due to the ever-growing gap between processor speed and memory speed. This renders similarity computation in mining tasks expensive due to the substantial amount of data transfer between processors and memory.

Recently, emerging non-volatile memory (NVM) devices like resistive RAM (ReRAM) [1], spin-transfer torque RAM (STT-RAM) and phase-change memory (PCM) enable *processing-in-memory* (PIM) [2], which is an efficient approach to diminish expensive data transfer (to processors) by directly processing the data stored in such devices. Among these candidates of NVM devices, ReRAM has superior characteristics such as lower read latency, higher data density and lower write energy. In the past decade, the industry (e.g., HP, Samsung, Toshiba [3]) has been developing ReRAM for supporting PIM.

In this paper, we investigate *how to accelerate similarity-based mining tasks by using ReRAM PIM without compromising the accuracy of results*. Our challenges stem from the following characteristics of ReRAM. First, ReRAM relies on crossbars for processing data, and it only supports specific operations (e.g., dot-product or vector-matrix multiplication) but not arbitrary operations. Second, the operands in ReRAM PIM can only be non-negative integers (with limited precision), due to the nature of analog computation in ReRAM crossbars. Third, the write endurance of ReRAM is limited compared to DRAM.

We are aware of prior works on utilizing ReRAM PIM in several application domains, e.g., neural network [4], graph computing [5], and DNA alignment [6]. However, they have not studied how to accelerate similarity computation on multi-dimensional data.

To tackle the aforementioned challenges, we propose a framework that exploits ReRAM PIM to accelerate a given similarity-based data mining algorithm, while preserving the accuracy of results.

• First, we conduct performance profiling on the similarity-based mining algorithm to identify its performance bottleneck (Section IV).

• Second, we present our solution in Section V. Our idea is to decompose a similarity (or its bound) function into different parts so that: (i) most of the computation can be quickly performed by using ReRAM PIM, and (ii) the remaining parts can be executed in the host processor (e.g., CPU) with little data transfer. We establish PIM-aware bound computation so that the accuracy of results won't be compromised.

Experimental results on real datasets show that our proposed method achieves up to 10.5x and 8.5x speedup for state-of-art kNN classification and k-means clustering algorithms, respectively (Section VI).

In the following, we first introduce the background of ReRAM PIM, and review similarity functions and similarity-based mining algorithms (Section II), then present an overview of our proposed framework (Section III).

II. Background

A. ReRAM Processing-in-memory

ReRAM is a promising candidate for future memory systems. A ReRAM cell can switch its state among different resistance levels, which can be used to represent a 1-7 bits integer. Each cell is connected by two orthogonal nanowires - wordline and bitline, and multiple wordlines and bitlines compose a *crossbar*. This crossbar structure leads to efficient computation in analog manner. By injecting voltages into wordlines, the currents measured at end of bitlines become the dot-product results of input voltages and conductance on cells. Figure 1 shows an example on a 3x3 2-bit crossbar.

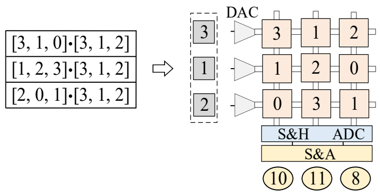


Fig. 1. Example of PIM dot-product operation on ReRAM crossbar.

ReRAM PIM can deal with high-precision data ($b > h$), where b denotes the bit size of data operand, and h denotes the bit precision of a ReRAM cell. A b-bit operand (multiplier) is segmented into multiple h-bit parts and then stored in adjacent cells at the same row. Similarly, multiplicands are segmented and converted to input voltages by every h bits. Dot-product is decomposed into several sub-operations, and the final result is obtained by *shifting* and *adding* the result of each sub-operation with circuit S&A [5]. Figure 2 depicts an example of processing 6-bit data on 2-bit cells. The decimal value '25' ('011001' in binary), is segmented to '1' ('01'), '2' ('10'), '1' ('01').

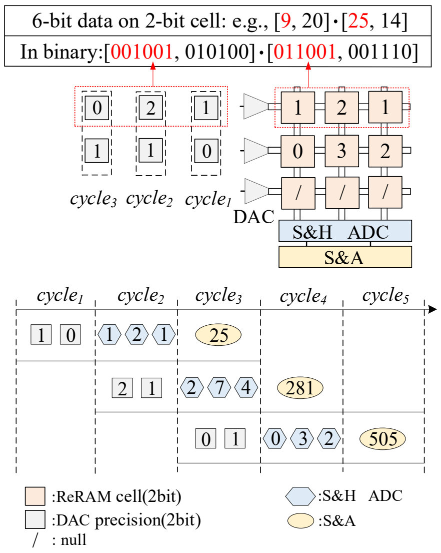

6-bit data on 2-bit cell: e.g., [9, 20]•[25, 14]
In binary:[001001, 010100]•[011001, 001110]
$cycle_3$ $cycle_2$ $cycle_1$
DAC
S&H ADC
S&A
$cycle_1$ $cycle_2$ $cycle_3$ $cycle_4$ $cycle_5$
25
281
505
:ReRAM cell(2bit)
:DAC precision(2bit)
/ : null
:S&H ADC
:S&A

Fig. 2. Example of pipeline of PIM dot-product operation on high-precision (i.e., $b{>}h$) data, example crossbar contains 3x3 2-bit cells. S&H is *sample and hold* circuit, S&A is *shift and add* circuit. DAC (ADC) is *digital (analog)-to-analog (digital)* converter.

ReRAM PIM can also compute dot-product on high-dimensional data, when the dimensionality exceeds the crossbar size. A crossbar is typically smaller than 1024×1024 [1]. Given a crossbar with the size of $m{\times}m$, a d-dimensional vector can be decomposed to a group of m-dimensional vectors, each of which is mapped to a crossbar (denoted as '*data crossbar*'). Then each data crossbar provides an output as partial results, and a crossbar (denoted as '*gather crossbar*') storing all-ones vector $\mathbf{e}{=}[1,...,1]$ is employed to sum up partial results vertically. Figure 3 presents an example of handling 6-dimensional vector on a 3x3 crossbar.

The high performance of ReRAM PIM relies on two features: massive parallelism and reduction of data transfer. First, massive crossbars compose ReRAM, and each crossbar serves as a processing unit to concurrently compute multiple data in a fashion similar to SIMD. Second, data is processed without

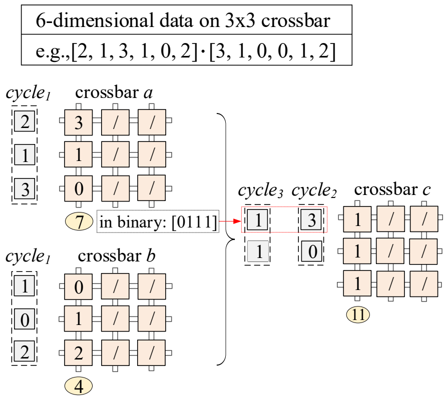

6-dimensional data on 3x3 crossbar
e.g.,[2, 1, 3, 1, 0, 2]•[3, 1, 0, 0, 1, 2]
$cycle_1$ crossbar a
7
in binary: [0111]
$cycle_1$ crossbar b
4
$cycle_3$ $cycle_2$ crossbar c
11

Fig. 3. Example of PIM dot-product operation on high-dimensional (i.e., $d{>}m$) data, example crossbar contains 3x3 2-bit cells (omit peripheral circuits for simplicity).

being moved to CPU. However, due to the analog computation on crossbars, ReRAM supports only dot-product operation on non-negative integer vectors but not arbitrary operations. Prior works [4], [6], [7] proposed several approaches to tackle the constrains, such as adding customized units to support more computing functionalities, and using fixed-point numbers to approximate floating-point values. Nevertheless, these approaches either can only be used for specific workloads, or cause the precision loss of results, which cannot be directly adopted to similarity-based mining tasks.

B. Similarity Measure

The similarity between high-dimensional objects is generally measured by distance. We focus on three distances: squared Euclidean distance (ED), cosine similarity (CS), and Pearson correlation coefficient (PCC) [8]. Let p denote a d-dimensional vector in dataset D. p_i is the value on i-th dimension and takes b-bit. N is the number of vectors in D. $dist(p, q)$ is the distance between vector p and q, such as $ED(p, q)$. The computation of these distances relies on simple arithmetic operations, yet triggers $O(d)$ cost of data transfer, which becomes the major bottleneck for many workloads.

C. Similarity-based Data Mining Algorithms

Similarity computation is an essential subroutine in many mining tasks. Especially, kNN classification and k-means clustering are two of the most widely used such tasks. Given an object q, kNN identifies k objects nearest to q [8]. While index structures suffer from "the curse of dimensionality", the distance bounds become an alternative to speedup kNN. Unpromising candidates are pruned by the bounds, so as to reduce calling expensive exact computation, which is known as the *filtering-and-refinement* paradigm. Table 1 shows representative distance bounds, where prefix LB denotes lower bounds of exact distance that satisfies $LB(p, q){\le}dist(p, q)$. The bounds adopt dimensionality reduction techniques, such as dividing d-dimensional vector into l segments with equal length d' (e.g., $l \cdot d' = d$).

k-means groups the nearest objects into k clusters [9]. ED is the most popular distance for k-means. Despite its age, Lloyd's algorithm [10] has shown efficiency of converging in a small number of iterations to near-optimal solution. Elkan [11], Drake [12], Yinyang [13] are the representative techniques to optimize Lloyd's algorithm. The common idea is to use triangle inequality to reduce the distance computation between data objects and cluster centers.

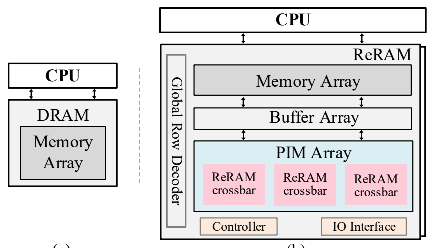
Table 1
REPRESENTATIVE BOUNDS FOR kNN CLASSIFICATION: LB_{OST} [14], LB_{SM} [15], LB_{FNN} [16], AND UB_{part} [17].

Symbol	Equation	Dis.
$LB_{OST}(p,q)$	$\sum_{i=1}^{d'}(p_i-q_i)^2+(\sqrt{\sum_{i=d'+1}^{d} p_i^2}-\sqrt{\sum_{i=d'+1}^{d} q_i^2})^2$	*ED*
$LB_{SM}(p,q)$	$l\cdot\sum_{i=1}^{d'}(\mu(\hat{p_i})-\mu(\hat{p_i}))^2$	*ED*
$LB_{FNN}(p,q)$	$l\cdot\sum_{i=1}^{d'}((\mu(\hat{p_i})-\mu(\hat{q_i}))^2+(\sigma(\hat{p_i})-\sigma(\hat{q_i}))^2)$	*ED*
$UB_{part}(p,q)$	$\sum_{i=1}^{d'} p_i q_i+\sqrt{\sum_{i=d'+1}^{d} p_i^2}\sqrt{\sum_{i=d'+1}^{d} q_i^2}$	*PCC* *CS*

III. OVERVIEW

A. PIM Architecture

Figure 4 depicts an overview of heterogeneous architecture with ReRAM PIM, functionalities of storage and processing are integrated on main memory. Each memory bank contains three parts: *memory array*, *buffer array*, and *PIM array*. Memory array works for storage, playing the same role as in traditional architecture to exchange data with host processor. PIM array is composed of multiple crossbars that are capable of processing the resided data. Next to PIM array, there is buffer array used to cache PIM results, as the massive parallelism of PIM produces abundant results concurrently. This simple architecture is widely adopted in prior works [2].



(a) (b)

Fig. 4. Overview of conventional architecture (a) and ReRAM-based processing-in-memory architecture (b).

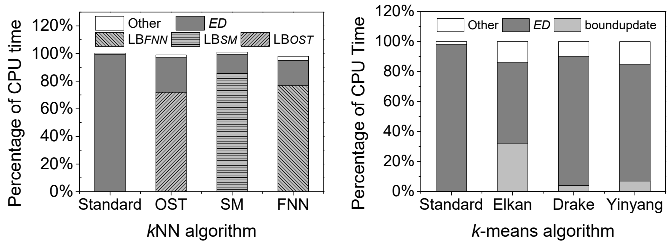
B. Systematic Framework

ReRAM PIM offers an opportunity of reducing data transfer from memory to processor. However, it does not support arbitrary computation. We propose a framework to make a similarity-based mining algorithm aware of characteristics of ReRAM PIM. The limited functionality of PIM makes it infeasible to run the entire algorithm. We use PIM for the similarity computation, and host processor (e.g., CPU) to coordinate the remaining steps. Given an algorithm, the first step is to conduct *performance profiling*, through which we identify the function causing the major bottleneck (Section IV). If the function is a similarity or bound function feasible to expose most computational task as dot-product, we define it as *PIM-aware function* that can enjoy offloading computation to PIM (Section V-A). We next present *PIM-aware bound function* based on non-negative integers, which helps to prune unpromising candidates and guarantee correct result (Section V-B). PIM array typically has limited space, which might be infeasible to accommodate the entire dataset. We then compress the dataset based on given hardware to avoid re-programming the crossbars (Section V-C).

IV. ALGORITHM PROFILING

Profiling an algorithm by functions helps us identify the function causing performance bottleneck. The execution time of an algorithm can be decomposed to components spent on each function and time T_{other} caused by all other operations such as condition check. Assume the algorithm contains t functions f_1, f_2, ..., f_t, and the time spent on each function is T_{f_1}, T_{f_2}, ..., T_{f_t} respectively. Then total execution time T_{total} of algorithm is T_{total}=$\sum_{i=1}^{t} T_{f_i}+T_{other}$.

Figure 5 depicts the execution time breakdown of kNN and k-means algorithms. 'Standard' denotes the linear scan method for kNN, and Lloyd's algorithm for k-means, respectively. FNN [16], OST [14], and SM [15] are state-of-art kNN algorithms, which employ the bounds shown in Table 1. Figure 5(a) shows that for kNN, calculation of *ED* dominates the execution time for Standard, and bound functions such as LB_{FNN} incur the majority (72%-86%) of total time for other algorithms. Figure 5(b) shows that the calculation of *ED* takes 52-96% of the execution time for k-means algorithms. Since the performance bottleneck is caused by either distance computation or bound computation, we plan to accelerate these functions by using PIM.



(a) MSD dataset , k=10 (b) NUS-WIDE dataset, k=64

Fig. 5. Execution time breakdown of representative kNN (a) and k-means algorithms (b).

V. ACCELERATING ALGORITHM WITH PIM

In this section. we present how to compute the distance and bound functions with PIM by addressing the limitations.

A. PIM-aware Function Decomposition

We identify a similarity or bound function as *PIM-aware function* if it can be decomposed to expose most computation as dot-product. Recall that ReRAM crossbars support specific operations (e.g., dot-product) but not arbitrary operations. Fortunately, it is possible to decompose a function into two parts: (i) sub-operations that can be processed by PIM, and (ii) sub-operations that can be pre-computed. PIM readily processes the vector data at online stage, the results of which are merged with pre-computed data in host processor for the final result. In this way, we decompose similarity or bound function $F(p,q)$ into PIM-aware format as follows:

$$F(p,q)=G(\Phi(p),\Phi(q),p\cdot q) \tag{1}$$

• $\Phi(p)$ takes a vector p from a dataset D as input and returns a fixed-size output. This function can be computed at

Table 2
PIM-AWARE DECOMPOSITION OF SIMILARITY FUNCTION AND BOUND FUNCTION.

	Offline	Online		
Function	$\Phi(p)$	$\Phi(q)$	$p{\cdot}q$	G
ED	$\sum_{i=1}^{d} p_i^2$	$\sum_{i=1}^{d} q_i^2$	$\sum_{i=1}^{d} p_i q_i$	$\Phi(p) + \Phi(q) - 2{\cdot}p{\cdot}q$
CS	$\sqrt{\sum_{i=1}^{d} p_i^2}$	$\sqrt{\sum_{i=1}^{d} q_i^2}$	$\sum_{i=1}^{d} p_i q_i$	$\frac{p\cdot q}{\Phi(p)\Phi(q)}$
PCC	Φ_a:$\sqrt{d\sum_{i=1}^{d} p_i^2 - (\sum_{i=1}^{d} p_i)^2}$	Φ_a:$\sqrt{d\sum_{i=1}^{d} q_i^2 - (\sum_{i=1}^{d} q_i)^2}$	$\sum_{i=1}^{d} p_i q_i$	$\frac{d\cdot p\cdot q - \Phi_b(p)\Phi_b(q)}{\Phi_a(p)\Phi_a(q)}$
	Φ_b: $\sum_{i=1}^{d} p_i$	Φ_b:$\sum_{i=1}^{d} q_i$		
LB_{FNN}	$l \cdot \sum_{i=1}^{d'}(\mu(\hat{p_i})^2 + \sigma(\hat{p_i})^2)$	$l \cdot \sum_{i=1}^{d'}(\mu(\hat{q_i})^2 + \sigma(\hat{q_i})^2)$	$\mu(\hat{p})\cdot\mu(\hat{q})$:$\sum_{i=1}^{d'} \mu(\hat{p_i})\mu(\hat{q_i})$	$\Phi(p)+\Phi(q)-2l\cdot\mu(\hat{p})\cdot\mu(\hat{q})$ $-2l\cdot\sigma(\hat{p})\cdot\sigma(\hat{q})$
			$\sigma(\hat{p})\cdot\sigma(\hat{q})$:$\sum_{i=1}^{d'} \sigma(\hat{p_i})\sigma(\hat{q_i})$	

offline stage. The same function $\Phi(q)$ can be applied on a given vector q involving applications, such as query object in kNN. It suffices to evaluate $\Phi(q)$ once. This can be computed in the host processor at the online stage.

• The dot-product operation $p \cdot q$ can be computed on PIM, which requires only constant data transfer cost.

• The function G is used to combine $\Phi(p)$, $\Phi(q)$ and $p \cdot q$ into the final result of $F(p, q)$ at online stage. This function can be calculated in host processor because of the constant time complexity.

ED is a PIM-aware function as it can be rewritten as follows:

$$\underbrace{ED(p,q)}_{G} = \underbrace{\sum_{i=1}^{d} p_i^2}_{\Phi(p)} + \underbrace{\sum_{i=1}^{d} q_i^2}_{\Phi(q)} - 2\underbrace{\sum_{i=1}^{d} p_i \cdot q_i}_{p\cdot q} \tag{2}$$

Indeed, *CS*, *PCC*, and the bound functions in Table 1 are also PIM-aware functions, because they can be decomposed as shown in Table 2. Note that Table 2 depicts some examples of corresponding function G, and the others are omitted due to space constraints. PIM-aware function can enjoy significant reduction of data transfer. For example, $ED(p, q)$ on d-dimensional vectors demands transferring $d{\cdot}b$ bits between memory and CPU on conventional architectures. In contrast, computation on the vectors is done by pre-processing or PIM in G, which reduces the data transfer to $3{\cdot}b$ bits.

B. PIM-aware Bound Computation

PIM can exactly compute on non-negative integer data. However, the similarity or bound functions such as *ED* typically operate on floating-point vectors, PIM cannot directly compute the exact value of the functions. The correct results of algorithm thus cannot be guaranteed. To tackle this, we utilize PIM to compute lower/upper bounds of distance functions, which still benefits from the significant reduction on data transfer. The lower/upper bounds are used to prune unpromising objects following filter-and-refinement strategy, which can guarantee the correct results.

Given dataset D, we initially normalize the floating-point values to be range of [0, 1]. Form here, scalar p_i is non-negative value within [0, 1]. We then enlarge p_i by multiplying constant α as scaling factor, and truncate the integer part $\lfloor \bar{p_i} \rfloor$. Then we have a vector $\lfloor \bar{p} \rfloor$ with only non-negative integers:

$$\bar{p_i} = p_i \cdot \alpha \tag{3}$$

$$\lfloor \bar{p} \rfloor = (\lfloor \bar{p_1} \rfloor, \lfloor \bar{p_2} \rfloor, ..., \lfloor \bar{p_d} \rfloor) \tag{4}$$

We then propose *PIM-aware bound function*, which serves as a bound of PIM-aware function. The involved dot-product operations just need to deal with non-negative integer vectors. As discussed in Section V-A, *ED*, *CS*, *PCC*, and bound functions in Table 1 are PIM-aware functions, each of which has the corresponding PIM-aware bound. We only present the PIM-aware bound for *ED* - LB_{PIM-ED} in Theorem 1, for lack of space. The PIM-aware bounds for other (bound) functions such as $LB_{PIM-FNN}$ for LB_{FNN} are omitted.

Theorem 1. *Squared Euclidean distance of two d-dimensional vectors p and q has a lower bound:*

$$LB_{PIM-ED}(p,q) = \frac{1}{\alpha^2}(\Phi(\bar{p}) + \Phi(\bar{q}) - 2\cdot\lfloor\bar{p}\rfloor\cdot\lfloor\bar{q}\rfloor - 2d) \tag{5}$$

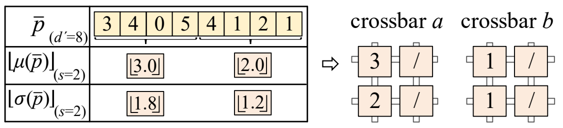
where $\Phi(\bar{p}) = \sum_{i=1}^{d} \bar{p_i}^2 - 2\sum_{i=1}^{d} \lfloor \bar{p_i} \rfloor$(resp. $\Phi(\bar{q})$), and $\lfloor\bar{p}\rfloor\cdot\lfloor\bar{q}\rfloor$ $=\sum_{i=1}^{d} \lfloor\bar{p_i}\rfloor\lfloor\bar{q_i}\rfloor$. Here, d is number of dimensions, and $\bar{p_i}$ is the non-negative floating-point value normalized from p_i with α, $\lfloor\bar{p_i}\rfloor$ is integer part of $\bar{p_i}$.

At offline stage, the value of $\Phi(\bar{p})$ and vector $\lfloor\bar{p}\rfloor$ are pre-computed, then stored on memory array and programmed on PIM array, respectively. During online stage, after receiving vector q, we calculate $\Phi(\bar{q})$ and $\lfloor\bar{q}\rfloor$ once. After PIM generates $\lfloor\bar{p}\rfloor \cdot \lfloor\bar{q}\rfloor$, only the pre-computed value of $\Phi(\bar{p})$ and value of $\lfloor\bar{p}\rfloor \cdot \lfloor\bar{q}\rfloor$ are transferred into CPU. Finally, the result of $LB_{PIM-ED}(p, q)$ is cheaply obtained after simple adding and subtracting operations.

C. PIM Memory Management

PIM-aware bound computation enjoys slight data transfer, but demands to store integer vectors such as $\lfloor\bar{p}\rfloor$ on crossbars, occupying the space of $N{\cdot}d{\cdot}b$ bits. Whereas, the typical capacity of contemporary PIM array is only 2GB [2], [7]. PIM array may not have sufficient capacity to accommodate the entire dataset. The simple solution is to divide the dataset into multiple small parts, and each time the crossbars are re-programmed with one part for processing. However, due to the limited write endurance of ReRAM, we should avoid re-programming crossbars. Hence, we propose to compress the dataset based on a given capacity.

The bound functions in Table 1 employ dimensionality reduction to effectively reduce memory usage. Apparently, the techniques can be adopted for PIM-aware bound functions to decrease the dimensionality from d (d') to s, by which the space cost is adjusted to $N{\cdot}s{\cdot}b$ bits, where s is the reduced dimensionality. Figure 6 depicts an example of computing $LB_{PIM\text{–}ED}$ on the compressed s-dimensional vector by adopting the dimensionality reduction of LB_{FNN}.



$\overline{p}_{(d'=8)}$
3 4 0 5 4 1 2 1
$\lfloor\mu(\overline{p})\rfloor_{(s=2)}$
3.0
2.0
$\lfloor\sigma(\overline{p})\rfloor_{(s=2)}$
1.8
1.2
crossbar a
crossbar b
3
2
1
1

Fig. 6. Example of reducing the dimension of vector from 8 to 4 (i.e., 2+2) for computing PIM-aware bound.

Theorem 2 establishes the condition of the dimensionality s so that the dataset can fit in PIM array and be processed in the right manner. It suffices to find the maximum value of s such that the approximation of PIM-aware bound obtains the highest possible closeness to the exact value.

Theorem 2. *Given hardware PIM array, and dataset having N d-dimensional vectors, the dimensionality s of compressed vectors is chosen as following conditions:*

$$\textit{Maximize: } s \tag{6}$$
$$\textit{subject to: } \begin{cases} n_{data} \le C & s \le m \\ n_{data} + n_{gather} \le C & s > m \end{cases}$$

where $n_{data} = \frac{N\cdot b\cdot s}{m^2\cdot h}$ is the number of crossbars serving as data crossbar, and $n_{gather} = \frac{N\cdot b}{m\cdot h}\sum_{i=2}^{\sqrt[m]{s}} \frac{s}{m^i}$ is the number of crossbars serving as gather crossbar. C is the number of crossbars in PIM array. Each crossbar contains $m\times m$ cells in h-bit precision, and b is the bit size of each operand.

We omit the detailed proof of Theorem 2 for lack of space. At offline stage, given PIM hardware and a dataset, s, n_{data} and n_{gather} are tuned by using Theorem 2. Then vector such as $\lfloor\overline{p}\rfloor$, and all-ones vector $\mathbf{e}$ are programmed to data crossbars and gather crossbars respectively.

VI. Evaluation

A. Experimental Setup

As commercial ReRAM PIM device is still not available, like prior NVM PIM research [4], [18], we resort to simulation. Specifically, we combine two simulators, a highly-flexible memory simulator NVSim [19] and a system-level tool Quartz [20], for accurately modeling PIM architecture. The configurations of ReRAM PIM and baseline architecture are illustrated in Table 3. The ReRAM-based memory has the same total size as DRAM in baseline platform, i.e., 16GB, in which 2GB is used as PIM array.

Table 3
The configuration of hardware platform.

CPU	Broadwell 2.10 GHz Intel Xeon E5-2620; Cache 1/2/3 : 32 KB/256KB/20MB;	
DRAM	16GB DIMM DDR4	
ReRAM-based memory	Memory Array	14GB ReRAM
	Buffer Array	16MB eDRAM
	PIM Array	2GB ReRAM
	Internal Bus	50GB/s
ReRAM crossbar	256×256 2-bit precision cells; read/write latency: 29.31/50.88ns;	

We measure the execution time of algorithms. The original algorithms are executed on real hardware platform. The execution time of our proposed PIM-optimized algorithms is measured by using NVSim and Quartz. NVSim estimates the time of PIM-involved processing executed on ReRAM-based memory. Quartz is to estimate the time of remaining non-PIM computation in CPU that requires data transfer from memory. The total execution time of a PIM-optimized algorithm is taken as the sum of execution time reported by NVSim and Quartz.

For kNN, we compare the algorithm Standard (i.e., linear scan), OST [14], SM [15], and FNN [16], to respective PIM-optimized algorithm (e.g., 'Standard-PIM'). For k-means, we compare the algorithm Standard [10], Elkan [11], Drake [12], and Yinyang [13], to respective PIM-optimized algorithm. For each algorithm, we identify the bottleneck function through profiling, and follow the techniques in Section V to offload computation of the function by processing its PIM-aware (bound) function. Table 4 lists the used real datasets. Original floating-point vectors are transformed to non-negative integer vectors with α as 10^6.

Table 4
Statistics of real datasets.

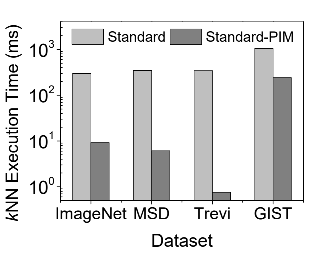
	Dataset	N	d	Size
kNN classification	ImageNet	2340173	150	3.5GB
	MSD	992272	420	2.9GB
	GIST	1000000	960	6.6GB
	Trevi	100000	4096	3.0GB
k-means clustering	NUS-WIDE	269648	500	280MB
	Enron	100000	1369	268MB

B. kNN Classification

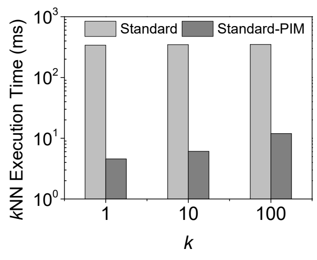
We first evaluate kNN PIM-optimized algorithms with varying datasets, k and distance functions in Figure 7. We set $k = 10$, distance ED as default setting, and MSD as default dataset. When PIM array is insufficient to process the dataset, we compute $LB_{PIM-FNN}$ serving as the bound of ED. Figure 7(a) shows that Standard-PIM achieves up to 453x speedup compared to Standard. Speedup becomes higher as the increase of data dimensionality d. Trevi occurs the highest speedup due to the significant reduction of data transfer: from $4096{\cdot}b$ to $3{\cdot}b$ bits for each distance computation. However, Standard-PIM shows slight speedup on GIST. This is because the data size of GIST exceeds PIM array capacity, and we compute $LB_{PIM\text{–}FNN}$ instead of $LB_{PIM\text{–}ED}$. $LB_{PIM\text{–}FNN}$ is based on LB_{FNN}, and LB_{FNN} shows weak pruning efficiency on GIST. For example, reducing the dimensionality to $d/4$ for LB_{FNN} provides average 71.3% approximation of exact distance on GIST, yet 95.4% on MSD.

Figure 7(b) depicts the execution time respect to different k. Standard-PIM yields 71.5x, 57.1x and 29.2x speedup compared to Standard respectively. The time ascends slightly with the increase of k due to more time spent on exact distance calculation for refining objects. We then study the impact of different distance functions. Figure 7(c) shows that the performance gaps between Standard-PIM and Standard on three distance functions are similar.

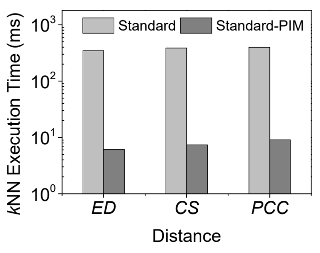
We proceed to compare OST, SM, and FNN, to their PIM-optimized algorithms respectively. Figure 7(d) shows that PIM contributes the significant improvement on all algorithms. Though the algorithms alleviate the data transfer overhead



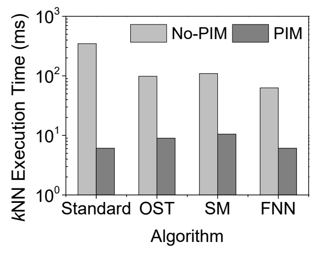
(a) Vary dataset



(b) Vary k



(c) Vary distance function



(d) Vary algorithm

Fig. 7. kNN classification execution time.

Table 5
EXECUTION TIME ON k-MEANS CLUSTERING.

Dataset	k-means execution time/iteration (ms)								
	k	Standard	Elkan	Drake	Yinyang	Standard-PIM	Elkan-PIM	Drake-PIM	Yinyang-PIM
NUS-WIDE	4	153.6	39.8	61.1	56.5	95.0	**37.3**	52.0	54.4
	64	1437.9	119.7	605.8	528.5	218.8	**94.8**	130.4	177.8
	256	5636.9	351.1	2451.2	1778.4	463.9	**253.7**	305.1	364.9
	1024	22273.1	1569.2	10545.3	5957.2	1274.7	**1201.0**	1240.8	1210.5
Enron	4	156.6	20.1	54.8	62.1	49.5	**18.2**	43.6	47.0
	64	1259.6	96.9	447.4	382.5	141.7	**82.8**	85.5	86.9
	256	4764.5	179.6	252.5	759.7	287.7	**162.4**	180.9	187.6
	1024	18984.5	317.2	4102.9	879.6	566.9	**282.6**	486.5	507.3

by computing distance bounds, utilizing PIM to compute the bounds further reduces the overhead effectively. On average, these state-of-art algorithms are 3.9x faster than Standard, and PIM further improves these algorithms by 10.5x speedup.

C. k-means Clustering

Table 5 shows the exeuction time of k-means algorithms and PIM-optimized ones with varying datasets and the number of centers. PIM is used to compute LB_{PIM-ED}, which contributes to filter most far apart centers. Note that though the overall speedup for k-means is not significant as kNN, recall that the goal of our work is to accelerate a given algorithm, rather than a certain application. Leveraging PIM yields the acceleration to each algorithm. Specifically, PIM gives consistent speedup for Standard, up to 33.4x. Elkan-PIM slightly outperforms Elkan. This is because *ED* calculation is not always the dominating task of Elkan. Updating original bounds often occupy up to 45% of total time. Besides, Drake-PIM achieves up to 8.5x speedup. Yinyang also enjoys the improvement caused by PIM. The significant speedup occurs on high-dimensional datasets, up to 4.9x.

VII. CONCLUSION

PIM is an efficient approach to reduce the substantial data transfer for similarity computation. We present a novel framework to accelerate a given similarity-based mining algorithm by using ReRAM PIM. We propose to identify PIM-aware function in the algorithm, and then offload most computation of the function into PIM. Our future work will design a space-friendly PIM scheme to deal with very large datasets.

ACKNOWLEDGMENT

This work was supported by grant GRF 152018/20E from Hong Kong RGC.

REFERENCES

[1] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design of cross-point metal-oxide reram emphasizing reliability and cost," in *ICCAD*, 2013, pp. 17–23.
[2] D. Fujiki, S. A. Mahlke, and R. Das, "In-memory data parallel processor," in *ASPLOS*, 2018, pp. 1–14.
[3] T. Liu, T. H. Yan, and et al, "A 130.7mm^2 2-layer 32gb reram memory device in 24nm technology," in *ISSCC*, 2013, pp. 210–211.
[4] 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 *ISCA*, 2016, pp. 27–39.
[5] L. Song, Y. Zhuo, X. Qian, H. H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in *HPCA*, 2018, pp. 531–543.
[6] W. Huangfu, S. Li, X. Hu, and Y. Xie, "RADAR: a 3d-reram based DNA alignment accelerator architecture," in *DAC*, 2018, pp. 59:1–59:6.
[7] L. Han, Z. Shen, D. Liu, Z. Shao, H. H. Huang, and T. Li, "A novel reram-based processing-in-memory architecture for graph traversal," *TOS*, vol. 14, no. 1, pp. 9:1–9:26, 2018.
[8] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.
[9] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.
[10] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–136, 1982.
[11] C. Elkan, "Using the triangle inequality to accelerate k-means," in *ICML*, 2003, pp. 147–153.
[12] J. Drake and G. Hamerly, "Accelerated k-means with adaptive distance bounds," in *5th NIPS workshop on optimization for machine learning*, vol. 8, 2012.
[13] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz, "Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup," in *ICML*, 2015, pp. 579–587.
[14] Y. Liaw, M. Leou, and C. Wu, "Fast exact k nearest neighbors search using an orthogonal search tree," *Pattern Recognition*, vol. 43, no. 6, pp. 2351–2358, 2010.
[15] B. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary lp norms," in *VLDB*, 2000, pp. 385–394.
[16] Y. Hwang, B. Han, and H. Ahn, "A fast nearest neighbor search algorithm by nonlinear embedding," in *CVPR*, 2012, pp. 3053–3060.
[17] C. Teflioudi, R. Gemulla, and O. Mykytiuk, "LEMP: fast retrieval of large entries in a matrix product," in *SIGMOD*, 2015, pp. 107–122.
[18] A. van Renen, V. Leis, A. Kemper, T. Neumann, T. Hashida, K. Oe, Y. Doi, L. Harada, and M. Sato, "Managing non-volatile memory in database systems," in *SIGMOD*, 2018, pp. 1541–1555.
[19] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.
[20] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li, "Quartz: A lightweight performance emulator for persistent memory software," in *Middleware*, 2015, pp. 37–49.