PIM-DH: ReRAM-based Processing-in-Memory Architecture for Deep Hashing Acceleration

Fangxin Liu[1,2], Wenbo Zhao[1], Yongbiao Chen[1], Zongwu Wang[1], Zhezhi He[1], Rui Yang[1]
Qidong Tang[1], Tao Yang[1], Cheng Zhuo[3], Li Jiang[1,2,4,*]
[1]Shanghai Jiao Tong University, [2]Shanghai Qi Zhi Institute, [3]Zhejiang University
[4]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University

ABSTRACT

Deep hashing has gained growing momentum in large-scale image retrieval. However, deep hashing is computation- and memory-intensive, which demands hardware acceleration. The unique process of hash sequence computation in deep hashing is non-trivial to accelerate due to the lack of an efficient compute primitive for Hamming distance calculation and ranking.

This paper proposes the first PIM-based scheme for deep hashing accelerator, namely PIM-DH. PIM-DH is supported by an algorithm and architecture co-design. The proposed algorithm seeks to compress the hash sequence to increase the retrieval efficiency by exploiting the hash code sparsity without accuracy loss. Further, we design a lightweight circuit to assist CAM to optimize hash computation efficiency. This design leads to an elegant extension of current PIM-based architectures for adapting to various hashing algorithms and arbitrary size of hash sequence induced by pruning. Compared to the state-of-the-art software framework running on Intel Xeon CPU and NVIDIA RTX2080 GPU, PIM-DH achieves an average 4.75E+03 speedup with 4.64E+05 energy reduction over CPU, 2.30E+02 speedup with 3.38E+04 energy reduction over GPU. Compared with PIM architecture CASCADE, PIM-DH can improve computing efficiency by 17.49× and energy efficiency by 41.38×.

1 INTRODUCTION

Large-scale image retrieval has attracted increasing attention due to its wide application in various scenarios (e.g., recommendation systems [12], search engines [18], etc.). For example, more than 1 billion images are uploaded every month on Facebook [7]. Therefore, the effective search of massive data becomes more critical, and nearest neighbor search, i.e., searching for the most similar data items, becomes an important research topic. However, the search efficiency of nearest neighbor search is limited by the dramatic increasing data volume, storage space, and computation complexity. To tackle this challenge, hashing-based methods have been proposed and achieved remarkable successes due to their accuracy and efficiency [2, 21]. It aims to learn a hash function that maps images in a high-dimensional pixel space to a low-dimensional Hamming space while maintaining their similarity in the pixel space [23].

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530575

Existing hashing-based image retrieval methods can be divided into shallow methods and deep learning-based methods (called deep hashing methods) in terms of adopted feature extraction mechanisms [2, 11]. Compared to shallow methods, deep hashing methods, which exploit the powerful ability of convolutional networks to capture features, significantly improve performances [23]. In this paper, we focus on deep hashing methods, whose computational paradigm consists of three phases, feature extraction phase, hash generation phase, and retrieval phase [1, 21]. 1) In the feature extraction phase, it learns to map data from the original space to the feature space through a deep convolutional neural network; 2) In the hash generation phase, it maps extracted features to hash sequences so that the generated hash sequence heavily depends on the extracted features; 3) In the retrieval phase, it computes the hash sequences with category correlations and ranks their Hamming distance, and outputs the closest category.

Nevertheless, deep hashing methods require considerable computational resources owing to the complicated nature of the feature extraction and Hamming distances calculation for large-scale data (e.g., image retrieval on the recommending platform in Taobao, which requires hash computations on 600 billion entries [17]). In order to boost the inference performance of deep hashing methods without reducing accuracy, systems (e.g., recommendation system in Facebook, Taobao) start to adopt efficient domain-specific hardware accelerators for at-scale retrieval [7]. The major performance bottlenecks come from the heavy use of dot-product in the feature extraction phase, massive number of searches in the retrieval phase [21, 23], and the frequent data movement between DRAM and processing elements (PEs) [15].

We identify processing-in-memory (PIM) as an effective architecture that can accelerate both the execution of vector-matrix multiplication and hash computation in deep hashing methods through performing the computation and search within memory macros [9, 15]. PIM dot product engine and PIM search engine can be efficiently implemented in Multiply-Accumulate (MAC) crossbar as well as Content Addressable Memory (CAM) crossbar using Resistive Random Access Memory (ReRAM) [19]. Previous ReRAM-based PIM accelerators have demonstrated their huge potential in energy-efficient vector-matrix multiplication [4, 15] and content-based search operation [3, 19]. Unfortunately, it is nontrivial to extend these architectures to the field of image retrieval. The main hurdle is that the PIM architecture stores the operands of hash computation (i.e., gallery hash sequences) to the CAM crossbar and binds these operands to the searching operation. Unlike the conventional Von-Neumann architecture, in which the PEs can handle the Hamming distance calculation and ranking, the PIM architecture can only efficiently search for the matching between two sequences and lacks the primitive to support hash calculation. Thus, the efficiency of hash computation in the PIM architecture is bounded by the length of the hash sequence and the searching mechanism. In summary, the extremely high search cost makes it challenging to accelerate deep hashing by the PIM-based design.

To combat these challenges, in this paper, we innovate a novel PIM-based scheme called PIM-DH to accelerate deep hashing for image retrieval tasks. We filter out the useless hash codes in the hash sequence inspired by weight pruning to improve search efficiency. In addition, we design the assisted circuit to optimize the efficiency of CAM matching for hash computation by using the leakage current latency mechanism. The main contributions of our work can be summarized as follows:

- As far as we know, this work is the first to exploit PIM to accelerate deep hashing for image retrieval tasks.
- We propose the hash sequence pruning to filter out redundant hash codes that have no contributions or even negative contributions to improve the resource utilization and reduce the complexity of hash computation.
- We design an execute-search dual-engine PIM-based architecture, including MAC compute engine, interface circuits and tailored CAM compute engine, to support flexible pruning method and optimize hash computation without incurring high extra costs.
- Experimental results of various datasets show that PIM-DH achieves up to 17.49× speedup and 41.38× energy efficiency improvement over the PIM-based accelerator.

2 BACKGROUND AND MOTIVATION

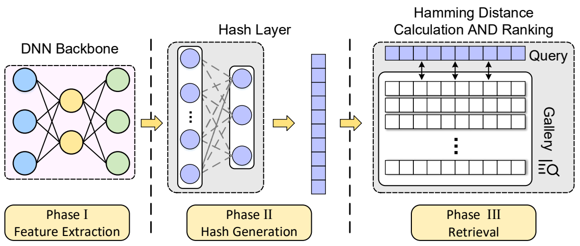

Figure 1: The process of deep hashing methods.

2.1 Deep Hashing Method

With the explosive growth of data, hashing methods have been drawing more attention [11, 23]. By reducing the dimensionality of high-dimensional feature matrix into low-dimensional, compact binary hash codes, hashing methods have advantages in retrieval speed and storage overhead that other methods cannot match [1, 2]. As shown in Fig. 1, deep hashing methods mainly consist of three phases: training the backbone neural network to extract high-dimensional features from the input data; mapping the extracted high-dimensional features into compact binary hash sequences; comparing the generated hash sequences with the gallery hash sequences, and ranking their Hamming distance to find the most similar category.

2.2 ReRAM-based PIM Designs

ReRAM is the emerging non-volatile memory with appealing properties of high density, fast read access, and low leakage power [4, 19]. ReRAM crossbar arrays support parallel in-memory MAC operations and ReRAM CAM for the match [10, 15, 20].

ReRAM-based MAC Crossbar. Existing ReRAM-based neural network accelerators utilize plenty of crossbar arrays as low-energy and high-speed dot-product engines [15, 20]. As shown in Fig. 2(a), ReRAM-based crossbars handle vector-matrix multiplication in the analog domain. First, the inputs are converted by the DACs into voltage pulses to drive the cells on the corresponding wordlines. Then, the currents generated in cells from the same bitline based on Kirchhoff's law are aggregated, representing the output of the MAC operation. Due to process limitations, each weight is partitioned into multiple cells to store, so the total current in each bitline corresponds to the partial sum of the product. Next, accumulated currents from the bitline are transferred to the ADC to convert the current amplitude (analog signal) into a digital signal to accumulate the partial sums further [4, 15].

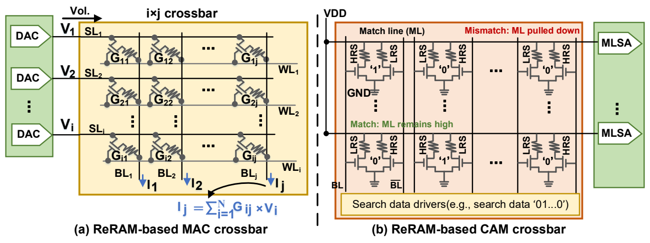

Figure 2: The overview of ReRAM-based crossbar.

ReRAM-based CAM Crossbar. CAM is often used in hardware implementation of in-memory computing for parallel search of large datasets because of its high speed and energy-efficiency [3, 19]. It compares the input search data with the stored data table and returns the matching data address. Compared with conventional 8-transistor SRAM-based Ternary CAM (TCAM), 2-Transistor-2-Resistor ReRAM-based TCAM has 6× higher density [19]. As shown in Fig. 2(b), ReRAM-based TCAM realizes bitwise XNOR-based search operations on each pair of cells by applying complementary bias voltages to the ReRAM devices [19]. When a row of stored data is matched, the connected sense amplifier (SA) with the row fires a matching signal, which would be captured for further processing. Binary CAM is the simplest CAM, which can search for data entirely consisting of '1's and '0's. TCAM allows a third match state of 'X', which means do not care. Data bit '1' consists of the left cell in High Resistance State (HRS) and the right cell in Low Resistance State (LRS), while the data bit '0' is the opposite, and data bit 'X' consists of both cells in HRS.

3 MOTIVATION AND KEY IDEA

Even though the prior ReRAM-based PIM designs have explored DNN accelerations and CAM-based match applications, none of these works can directly accelerate deep hashing methods for efficient image retrieval due to the intrinsic challenges as follows:

Massive number of searches. Hamming distance calculation involves multiple searches for ReRAM CAM. *This is because the matching principle of ReRAM CAM on the leakage current mechanism can check only whether two contents are equal or not (i.e., the equality comparison of the query hash sequence and the gallery hash sequence) [3, 8, 19].* Suppose the length of query hash sequence is L, in the worst case, $2^L - 1$ matches on ReRAM CAM in a bit-by-bit masked way are required for the Hamming distance calculation, which is a significant overhead.

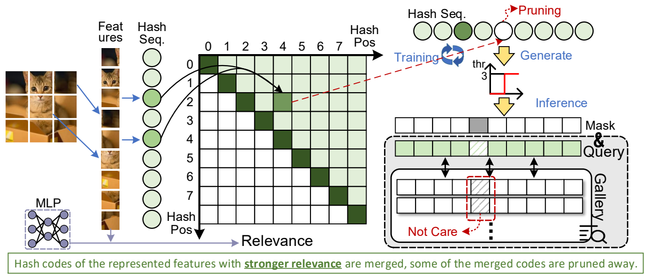

Figure 3: Relation-aware hash sequence pruning.

Extreme CAM overhead. The gallery hash sequences stored in the ReRAM CAM are determined by the length of hash sequences. A typical image retrieval application needs hash sequences of the length of 64 or 128 [1, 2]. For example, ILSVRC2012 dataset [6] contains 10,000,000 labeled images, requiring $\approx$ 153MB of CAM overhead with the 128-bit hash sequence.

We observe that the length of the hash sequence determines both the retrieval efficiency and the CAM overhead. Therefore, in this work, we propose PIM-DH, a first ReRAM-based deep hashing accelerator with algorithm/hardware co-design optimizations. A novel relevance-ware pruning method is proposed to reduce hash codes required in Hamming distance calculation, enabling little redundancy among the features represented by the hash codes. This pruning method consists of offline structural pruning and on-the-fly non-structural pruning, which reduce the CAM occupancy of gallery hamming sequence and the latency of hamming distance calculation, respectively. Crucially, our lightweight assisted logic circuit fully leverages the leakage current latency match of ReRAM CAM to perform Hamming distance, thereby reducing the number of matches, resulting in less latency and energy consumption of Hamming distance calculation.

4 RELEVANCE-AWARE PRUNING

We exploit the fine-grained hash code sparsity and prune hash codes that are useless or even play a negative role in the retrieval phase. The pruning method has two phases: 1) offline structural pruning to reduce the length of gallery hash sequence; 2) on-the-fly relation-aware pruning to filter out redundant hash codes of query hash sequence. We aim to boost retrieval efficiency by removing the hash codes with relevance overlap of features in the hash sequence $\boldsymbol{h}_{\mathrm{k}} = \{h_0, ..., h_L\};\ h_i \in \{0, 1\}$. We use a two-layer Multi-Layer Perception (MLP) module to capture the relevance among the hash codes represented by the feature, which can be described as:

$$R_{i,j} = MLP([\mathcal{F}_i \parallel \mathcal{F}_j]),\ \forall\ 0 \leq i, j \leq L, i \neq j \tag{1}$$

where $\parallel$ indicates the features represented by the hash code h_{i} are concatenated with the features represented by the hash code h_{j}. We show the overall logic of our algorithm in Fig. 3. Based on the relevance metric, we exploit hash code sparsity in two phases.

Offline Crossbar-aware Pruning. A warm-up method is to combine the structural sparsity with crossbar size. We first apply the structural pruning on the gallery hash sequences. After that, we store the gallery hash sequences on the CAM with the exception of the pruned hash codes in the hash sequence. As the length of hash sequences is shorter, multiple hash sequences can be placed in the same crossbar-row, increasing resource utilization. However, this results in an increase in the number of searches required, which will be discussed in Section 6.2.

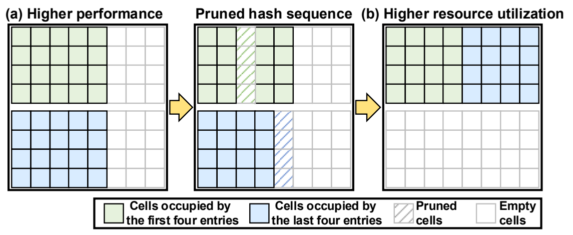
(a) Higher performance
Pruned hash sequence
(b) Higher resource utilization
Cells occupied by the first four entries
Cells occupied by the last four entries
Pruned cells
Empty cells

Figure 4: The gallery hash sequence mapping on the crossbar array.

We find that the length of the hash sequence and the size of the selected crossbar affect the selection of the pruning rate. Fig. 4 shows an example of the above process. We summarize three rules: 1) Suppose the length of the hash sequence is larger than the crossbar size. Here, we take the crossbar size as a strict limit, i.e., the length of pruned hash sequence will be less than or equal to an integer multiple of the crossbar size. Suppose the hash sequence is smaller than the crossbar size. 2) To achieve better performance, one hash sequence is mapped to a row of the crossbar; 3) To achieve higher resource utilization, multiple hash sequences are mapped to one row of the crossbar.

On-the-fly Relation-aware Pruning. Suppose we need L hash codes to represent the query hash sequence; the goal is to represent the whole query hash sequence with fewer hash codes while guaranteeing the retrieval accuracy of images. Since different input values may have different impacts on the final accuracy [22], we assume that each hash code in the query hash sequence contributes differently to the category classification. Thus, we want to set a mask sequence $\boldsymbol{m}_{\mathrm{k}} = \{m_0, ..., m_L\};\ m_i \in \{0, 1\}$ to mask some hash codes with redundant information. The length of the mask sequence is equal to the length of the original query hash sequence L and masking bits '0' identify the position of pruned in the original hash sequence.

We integrate the training process to evolve hash code sparsity by enforcing relevance-wise restrictions at every training iteration. In the forward pass, the relevance among the hash codes is made as the MLP output. Then, the hash sequence is sparsified based on the relevances, and the hash computations are carried out with the sparse version of the hash sequence. The sparsified hash codes indicate that these positions in the hash sequence do not need to be compared. The mask sequence $\boldsymbol{m}_{\mathrm{k}}$ is generated in every forward pass by identifying the maximum overlap relevance of features. The binary mask of $\boldsymbol{m}_{\mathrm{k}}$ element-wisely is applied on the query hash sequence for hash codes selection in the retrieval phase.

During the training, the r-percentile of relevance of features, which exceeds $r * L$ of them, is recorded. The average value of features of all these r-percentiles is denoted as thr, which can eliminate the outputs in the top r portion based on the features with the larger overlap relevance against each other. Specifically, *we aim to enable the features represented by the remaining hash codes to have little relevance to each other.* The generated mask sequence also plays an important role in the backward pass. The gradients for the weights connected to without masked hash codes are directly back-propagated, and the gradient for the weights corresponding to the features represented by masked hash codes is multiplied by a scaling factor:

$$\frac{\partial}{\partial \boldsymbol{w}} = \boldsymbol{m}_{\mathrm{k}} \odot \frac{\partial}{\partial \boldsymbol{w}} + (\boldsymbol{I} - \boldsymbol{m}_{\mathrm{k}}) \odot \frac{\partial}{\partial \boldsymbol{w}} \alpha \tag{2}$$

where $\boldsymbol{I}$ is the vector, the element's value is set to 1 and the size is equal to the mask $\boldsymbol{m}_k$. Using a smaller value as the scaling factor α (e.g., 0.1) usually shows superior performance in practice compared to propagating the gradient directly to the masked weights ($\alpha = 1$) or masking its gradient completely ($\alpha = 0$) [14]. We repeat the above training process until the model converges. Next, we conduct the on-the-fly pruning for inference and evaluate whether the retrieval accuracy can meet the expected requirement. If yes, the threshold are determined. Otherwise, we will repeat the above steps by halving the threshold. Through trial-and-error, the above process can always find the satisfactory values within a few iterations.

During the inference, if the feature value exceeds the threshold thr, the bit at this position of the mask sequence $\boldsymbol{m}_k$ is set to '1', which means this hash code represented features is important. As a result, we feed the pruned hash sequence $\boldsymbol{h}_k \cdot \boldsymbol{m}_k$ to the next retrieval phase, and the mask sequence $\boldsymbol{m}_{1:L}$ is integrated into the CAM compute engine.

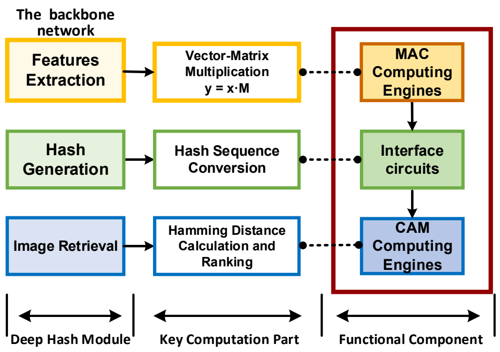

Figure 5: Overview of the PIM-DH for deep hashing methods.

5 PIM-DH ARCHITECTURE

This section begins with the present the overall architecture of PIM-DH and demonstrate the new data path. Afterward, we design lightweight circuits to extend the CAM to support efficient compute primitive for Hamming distance calculation and ranking.

5.1 Overall Architecture

As shown in Fig. 5, deep hashing methods consist of three main modules: feature extraction, hash generation, and image retrieval, where the main computational parts are vector-matrix multiplication, hash sequence conversion, and Hamming distance calculation and ranking. We design efficient architecture around all the above operations to support them, as shown in Fig. 6, which consists of three types of functional components.

Vector-Matrix Multiplication can be efficiently completed by **MAC Compute Engines (❶ in Fig. 6)**, consisting of ReRAM crossbars, SRAM-based input and output buffers, which supply inputs to and store outputs from the crossbars and peripheral circuits. MAC Compute Engines are responsible for executing the computations of fully connected and convolutional layers in the backbone network, where the dominant computation is the MAC operation.

Hash Sequence Conversion then compares the image signatures generated by feature extraction with the threshold to yield the binary hash sequence for image retrieval. This can be supported by **Interface Circuits (❷ in Fig. 6)**, which consists of logic circuits between the MAC compute engines and the CAM compute engines.

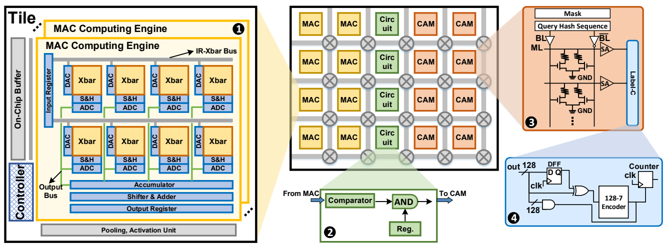

Figure 6: The architecture and circuits of the proposed PIM-DH.

It dynamically controls the mask generation to skip the redundant hash codes for feeding not care state to CAM compute engines. We first fetch inputs to the MAC compute engines, then, we need to convert the output (extracted features) of the MAC crossbar to the hash code in the query sequence, which specific steps are as follows: 1) the hash sequence is generated by comparing the result from the MAC compute engine with the threshold; 2) the query sequence is combined with the pruning threshold to adjust the mask sequence; 3) the "AND" operation is performed to generate the pruned query sequence and the corresponding mask for the subsequent image retrieval operation. After that, the pruned query sequence is fed to the CAM compute engines for querying the category.

Hamming Distance Calculation and Ranking can be efficiently processed by **CAM compute engines**, consisting of CAM crossbar (❸ in Fig. 6)) assisted with dedicated lightweight circuit (❹ in Fig. 6)), which serve as the searching engines to execute the hash computation of image retrieval efficiently. The main idea is to architect an extra circuit to capture the latency of leakage current when the mismatch happens among the query and gallery sequences. As a result, SIMSnn achieves the differential comparison than the equality comparison in the conventional designs [3, 19] to reduce the latency of Hamming distance calculation.

Multiple compute engines (denoted by MAC, CAM in Fig. 6)) and interface circuits are connected together in a mesh-based network while the data flow between different phases in a pipelined manner. The central controller orchestrates the flow of operation among these functional components.

5.2 CAM Search Mechanism Optimization

In this section, we present the implementation details of Hamming distance calculation in PIM-DH, and optimize the search mechanism to reduce the number of frequent searches for hash sequences. ❸ and ❹ in Fig. 6 illustrate the implementation of the searching process. We take full advantage of the execution mechanism of CAM to achieve the efficient search of hash sequences. Initially, we set the bits of the mask register induced by the hash sequence pruning. First, the mask register will activate the bits corresponding to '1' in the mask to perform the match operation between the query and gallery sequences. In this way, PIM-DH enables masking the column index according to the mask such that the match operation only compares unmasked hash codes in the sequence. This feature naturally supports our pruning on the query hash sequence.

We design the **assisted circuit** for ReRAM CAM to achieve efficient Hamming distance calculation and ranking. If some rows are tagged by the Label-C (❹), it means that the corresponding category is retrieved. Specifically, the closer the bits on the match line are to the given hash sequence, the more slowly the current on the match line leaks. Thus, we design the circuit for recording the

Table 1: PARAMETERS OF THE PIM-DH ARCHITECTURE.

Component	Configuration	Area (mm2×10-3)	Power (mW)
MAC Compute Engine			
MAC Crossbar	128 × 128 2-bit/cell number: 8	0.24	2.44
DACs	1-bit number:8 × 128	0.17	4.1
ADC	frequency:1.2GSps number:8 resolution:8-bit	9.6	15.9
S&H	number:8 × 128	0.023	0.0055
S+A	number:4	0.24	0.2
CAM Compute Engine			
CAM Crossbar	128 × 128, 1-bit/cell number: 8	0.34	2.84
Sense Amplifier	number: 8 × 128	2.35	5.42
Label-C Logic		0.033	0.014

latest leaked row number and counting corresponding cycles. "out" corresponds to the output of CAM, where each bit corresponds to whether the current cycle of that row produces output. "out" and the output of the previous cycle (recorded in the D Flip-Flop) perform an XNOR operation and are recorded in the D Flip-Flop (DFF) simultaneously. The XNOR operation results are fed to the encoder, and the corresponding row number is generated. Meanwhile, "out" is connected to AND gates, and the row number is output only when all rows have yielded their results. Here, we use a counter driven by the clock signal to record the corresponding cycle number. The AND gates output performs AND operation with the encoder output and is used as the enable signal for the counter to output recorded cycle.

6 EVALUATIONS

6.1 Experiment Setup and Benchmark

We build a simulator for the PIM architecture using the same MAC crossbar configuration as the ISAAC-like design [15]. The power consumption parameters of the CAM crossbars are obtained by performing SPICE simulations using the ReRAM model from [3] in 32nm technology. The read energy consumption and latency are 1.08pj and 29.31ns, respectively [3]. For the ADC and DAC, we use the model from [15]. We utilize CACTI [13] at 32nm technology to provide the power consumption and area of all memories (including eDRAM buffers, input register, output register, mask register and etc.). The designed circuits is modeled in Verilog RTL and synthesized using Synopsys Design Compiler [16] at 32nm technology. Table 1 shows the area and power parameters of the main components of our PIM-DH. The proposed pruning method retrain the backbone model after pruning for the threshold, and we adopt the ADMM algorithm for the retraining process [22]. During the retraining, we set the stochastic variation be $\sigma = 0.025$ to guarantee the retrieval accuracy and robustness of PIM-DH.

The datasets that we use to evaluate our proposed methods are two widely-studied image retrieval datasets: NUSWIDE [5], and ImageNet [6]. We evaluate DSH [11], DHN [23], CSQ [21], HashNet [2] on these datasets. For image retrieval, the length of the hash sequence usually used is 16 to 128. We adopt Mean Average Precision (mAP) as the metric, which is identical as used in [1, 2]. We compare the evaluation results of PIM-DH with three inference designs: (1) PyTorch on the Intel Xeon Silver 4108 CPU; (2) PyTorch on a NVIDIA RTX 2080 GPU, and (3) modified CASCADE [4] followed by the CAM for the hash computation in a naive manner.

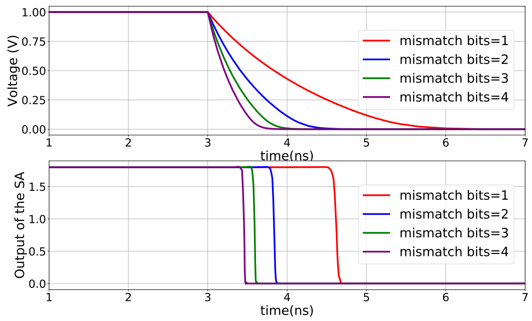

Figure 7: the voltage of the match line and the output of the SA versus mismatched bits.

6.2 Results and Analysis

Impact on the mismatched bits of CAM. Fig. 7 first shows the relationship between the number of mismatched bits and the voltage of match lines for the execution of the CAM. We can find the maximum number of mismatched bits that CAM has the potential to distinguish. All match lines are precharged with $1.0V$, LRS = $5K\Omega$, HRS = $50K\Omega$, and the number of mismatched bits is varied from 1 to 4. The voltage pull-down is attributed to the increment of mismatched bits on the same match line. PIM-DH records the time of discharge to identify the number of mismatched bits by the designed circuits and adjusts the frequency of the peripheral circuit to match the discharge speed as needed. Therefore, the higher the frequency of the peripheral circuit, the higher the number of mismatched bits that can be distinguished, and the lower the number of CAM searches performed. Naturally, this leads to higher overall system performance but lowers energy efficiency marginally.

Pruning Performance. Table 2 shows the comparison results on two datasets. We can observe that our proposed pruning method in PIM-DH can effectively remove useless, or even negative-role, hash codes in the hash sequence, resulting in better performance of the processed model. Specifically, performance boosts of $> 2\%$, $> 1\%$ in terms of mAP for NUSWIDE and ImageNet on average, respectively. Besides, the hash sequence length is reduced by our method, which can effectively increase the efficiency when performing CAM search for hash computation following.

Energy, Area Consumption and Performance. We set the CPU result as the baseline and compare the speedup and energy efficiency of deep hashing methods deployed on other platforms normalized to CPU, as shown in Fig. 8. We can find that PIM-DH

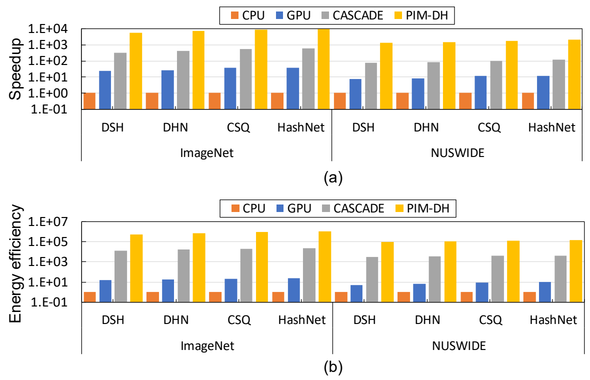

Figure 8: (a) Speedup and (b) Energy efficiency comparison with deep hashing methods (128-bit hash sequence) deployed on the various platforms (i.e., CPU, GPU and modified CASCADE).

Table 2: **Mean Average Precision (mAP) Comparison of Hamming Ranking with/without PIM-DH Under Different Hash Sequence Length.**

DataSets	Methods	Ori.	PIM-DH	Ori.	PIM-DH	Ori.	PIM-DH	Ori.	PIM-DH
		16 bits	12 bits	32 bits	21 bits	48 bits	30 bits	64 bits	39 bits
ImageNet	DSH [11] (CVPR)	0.4025	0.4137	0.4914	0.5077	0.5254	0.5371	0.5845	0.6038
	DHN [23] (AAAI)	0.4139	0.4315	0.4365	0.4536	0.468	0.4842	0.5018	0.5243
	HashNet [2] (ICCV)	0.3287	0.3472	0.5789	0.6003	0.6365	0.6621	0.6656	0.6948
NUSWIDE	DSH [11] (CVPR)	0.6338	0.6544	0.6507	0.6711	0.6664	0.6923	0.6856	0.7163
	DHN [23] (AAAI)	0.6471	0.6703	0.6725	0.6949	0.6981	0.7236	0.7027	0.7364
	HashNet [2] (ICCV)	0.6821	0.7062	0.6953	0.7231	0.7193	0.7506	0.7193	0.7558

achieves 4.75E+03 speedup and 4.64E+05 energy reduction over CPU, 2.30E+02 speedup and 3.38E+04 energy reduction over GPU on average, respectively, which is due to the energy efficiency characteristic of ReRAM crossbars. Besides, PIM-DH can also achieve an average 17.49× speedup and 41.38× energy reduction over PIM design. This is mainly because the process of hash computation is optimized by our dynamic pruning in Section 4 and the dedicated search mechanism in Section 5.2.

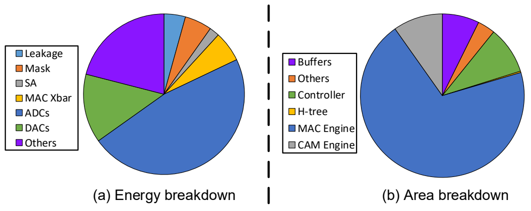

Figure 9: (a) Energy and (b) Area breakdown of PIM-DH.

We also investigate the energy consumption and area of each component in PIM-DH. Fig. 9(a) demonstrates the energy breakdown of the main compute engines (i.e., MAC compute engine and CAM compute engine) in the PIM-DH, where the MAC compute engine consumes most (67.92%) of energy (including MAC crossbar (MAC Xbar), ADCs, and DACs). The energy consumption for the hash computation makes full use of the search mechanism of CAM and the leakage mechanism of the designed circuits (Label-C mentioned in Section 5.2). Thus, the CAM compute engine occupies 11.37% energy is consumed by the precharging, leakage, mask register, etc. Other components cost 20.71% energy, consisting of the buffers and peripheral circuits for the hash generation. We can also observe that the area breakdown of PIM-DH in Fig. 9(b). The MAC compute engine (MAC engine) occupies 69.79% of the total area, where the H-tree costs 0.27% area, and the CAM compute engine (CAM engine) occupies 9.78% of the total area. The buffers, others (peripheral circuits), and controllers occupy 7.31%, 3.47%, and 9.38% area, respectively.

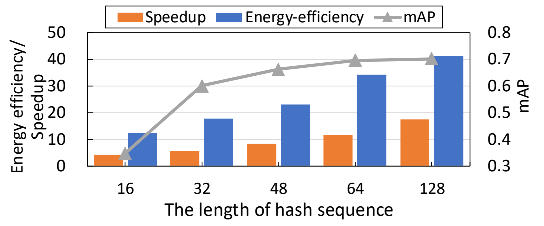

Figure 10: Analyzation of the length of hash sequence.

Varying length of hash sequence. Fig. 10 demonstrate the results on PIM-DH with different length of hash sequence. The results normalized to the modified CASCADE. HashNet with a short hash sequence shows the best performance on PIM-DH, while HashNet with a long hash sequence shows the most significant energy efficiency on PIM-DH. Such an improvement is mainly attributed to the gain of searching the gallery.

7 CONCLUSION

In this work, we propose PIM-DH, an execute-search dual-engine PIM architecture to accelerate the computation of deep hashing methods, including the hash sequence pruning method, peripheral circuits, and simple but effective PIM architecture. Our scheme not only exploits pruning methods to improve the retrieval efficiency but also leverages the characteristics of the CAM search mechanism to design peripheral circuits to optimize the overhead of hash computation. Experiments show that our PIM-DH achieves significant improvement in energy efficiency, performance, and accuracy.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (NSFC) (Grant No. 61834006, 62102257 and 62104140), the National Key Research and Development Program of China (2018YFB1403400), Lingang Laboratory "Qiusuo Distinguished Young Scholar" Open Research Fund (Grant LG-QS-202202-11) and Biren Technology–Shanghai Jiao Tong University Joint Laboratory Open Research Fund. We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University. Li Jiang is the corresponding author (ljiang_cs@sjtu.edu.cn).

REFERENCES

[1] Yue Cao et al. 2018. Deep cauchy hashing for hamming space retrieval. In *CVPR*.
[2] Z. Cao et al. 2017. Hashnet: Deep learning to hash by continuation. In *ICCV*.
[3] Nagadastagiri Challapalle et al. 2020. GaaS-X: Graph analytics accelerator supporting sparse data representation using crossbar architectures. In *ISCA*.
[4] Teyuh Chou et al. 2019. Cascade: Connecting rrams to extend analog dataflow in an end-to-end in-memory processing paradigm. In *MICRO*.
[5] Tat-Seng Chua et al. 2009. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*.
[6] Jia D. et al. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.
[7] Udit Gupta, Carole-Jean Wu, et al. 2020. The architectural implications of facebook's dnn-based personalized recommendation. In *HPCA*. IEEE.
[8] Huize Li et al. 2020. ReSQM: Accelerating Database Operations Using ReRAM-Based Content Addressable Memory. *TCAD* (2020).
[9] Fangxin Liu et al. 2021. Bit-Transformer: Transforming Bit-level Sparsity into Higher Preformance in ReRAM-based Accelerator. In *ICCAD*.
[10] Fangxin Liu et al. 2021. SME: ReRAM-based Sparse-Multiplication-Engine to Squeeze-Out Bit Sparsity of Neural Network. In *ICCD*.
[11] H. Liu et al. 2016. Deep supervised hashing for fast image retrieval. In *CVPR*.
[12] Julian McAuley et al. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*.
[13] Naveen Muralimanohar et al. 2007. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO*. IEEE.
[14] Elbruz Ozen et al. 2021. Evolving Complementary Sparsity Patterns for Hardware-Friendly Inference of Sparse DNNs. In *ICCAD*.
[15] Ali Shafiee et al. 2016. ISAAC: A CNN accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Comput. Archit. News* (2016).
[16] Synopsys. [Online]. https://www.synopsys.com/community/university-program/teaching-resources.html.
[17] Jizhe Wang, Pipei Huang, et al. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *KDD*.
[18] Lei Wu et al. 2012. Tag completion for image retrieval. *PAMI* (2012).
[19] Rui Yang et al. 2019. Ternary content-addressable memory with mos 2 transistors for massively parallel data search. *Nature Electronics* (2019).
[20] Xiaoxuan Yang et al. 2020. ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration. In *ICCAD*.
[21] Li Yuan et al. 2020. Central similarity quantization for efficient image and video retrieval. In *CVPR*.
[22] Tianyun Zhang et al. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*.
[23] H. Zhu et al. 2016. Deep hashing for efficient similarity retrieval. In *AAAI*.