ODLPIM: A Write-Optimized and Long-Lifetime ReRAM-Based Accelerator for Online Deep Learning

Heng Zhou, Bing Wu, Huan Cheng, Wei Zhao, Xueliang Wei, Jinpeng Liu, Dan Feng*, Wei Tong*
Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System,
Engineering Research Center of Data Storage Systems and Technology, (School of Computer Science and Technology,
Huazhong University of Science and Technology), Ministry of Education of China, Wuhan, China
{heng_zhou, wubin200, chenghuan, weiz, xueliang_wei, jinpengliu98, dfeng, Tongwei}@hust.edu.cn
*Corresponding author

***Abstract*—ReRAM-based Processing-In-Memory (PIM) architectures have demonstrated high energy efficiency and performance in deep neural network (DNN) acceleration. Most of the existing PIM accelerators for DNN focus on offline batch learning (OBL) which require the whole dataset to be available before training. However, in the real world, data instances arrive in sequential settings, and even the data pattern may change, which calls concept drift. OBL requires expensive retraining to solve concept drift, whereas online deep learning (ODL) is evidenced to be a better solution to keep the model evolving over streaming data. Unfortunately, when ODL optimizes models over a large-scale data stream in the PIM system, unbalanced writes are more severe than OBL, due to the heavier weight updates, resulting in the amplification of unbalanced writes and lifetime deterioration. In this work, we propose ODLPIM, an online deep learning PIM accelerator that extends the system lifetime through algorithm-hardware co-optimization. ODLPIM adopts a novel write-optimized parameter update (WARP) scheme that reduces the non-critical weight updates in hidden layers. Besides, a table-based inter-crossbar wear-leveling (TIWL) scheme is proposed and applied to the hardware controller to achieve wear-leveling between crossbars for lifetime improvement. Experiments show that WARP reduces weight updates on average to 15.25% and up to 24% compared to that without WARP, and eventually prolongs system lifetime on average to 9.65% and up to 26.81%, with a negligible rise in cumulative error rate (up to 0.31%). By combining WARP with TIWL, the lifetime of ODLPIM is improved by an average of 12.59× and up to 17.73×.**

***Index Terms*—online deep learning, processing-in-memory, write reduction, wear-leveling, lifetime extension**

I. Introduction

Many ReRAM-based DNN accelerators [1]–[3] have been proposed to address the memory wall issue in recent years. However, they focus on OBL which requires the entire dataset to be available before training. In fact, the real-world data instances arrive in a sequential setting and the target concepts may be drifting or evolving over time [4]. So OBL suffers from expensive retraining overhead to keep evolving when concept drift occurs, causing it to become increasingly restricted and poorly scalable. ODL has been proposed to solve this problem effectively and optimizes models over million-scale streaming data. However, if we directly apply PIM accelerators to ODL situations, the lifetime of the PIM system will drop dramatically according to our experiments:

- **Massive weight updates:** Optimizing DNN models over streaming data inevitably introduces enormous weight updates. The limited endurance of ReRAM makes it difficult to support the long-term training of DNNs with millions of iterations [5].
- **Unbalanced write:** Based on our observations, quantization drops small updates leading to an unbalanced write distribution of weight value. In addition, the different update frequencies of the most significant bits (MSB) and the least significant bits (LSB) of data result in unbalanced writes within the cells representing one value.

Although unbalanced write exists in both ODL and OBL, it is more serious in ODL. Because the scale of streaming data is often more than one million or even larger, ODL suffers from heavier weight updates, resulting in the amplification of unbalanced writes and lifetime deterioration. Wear-leveling is a straightforward solution to balance the writes of the PIM system. However, existing works, such as Row swapping (RS) [5], column swapping (CS) [6] and row-column swapping (RCS) [7], focus on wear-leveling within a crossbar (or algorithm-level matrix). Even though they prolong the lifetime of crossbars, the lifetime of the whole system still depends on the worst crossbar. Thus, a more desired option is to design a wear-leveling scheme that ensures balanced writes in the whole system to further improve the lifetime.

In this paper, we propose an online deep learning PIM accelerator called ODLPIM and design an algorithm-hardware co-optimization to prolong ODLPIM's lifetime. The main contributions of this paper include:

- We find that the Hedge Backpropagation algorithm of ODL shifts training attention by controlling the distribution of contribution rates of all layers. Therefore at the algorithm level, WARP is proposed to reduce the weight updates of layers with a low contribution.
- We propose an orthogonal inter-crossbar wear-leveling scheme, called TIWL, which remaps the "hot" logical arrays to the less-worn physical crossbars and extends

978-3-9819263-7-8/DATE23/© 2023 EDAA

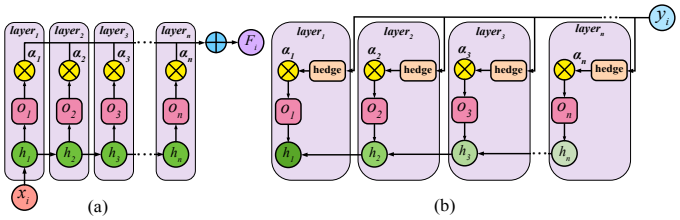


Fig. 1. (a) Forward stage of HBP; (b) Backward propagation of HBP.

the lifetime of ODLPIM at the hardware level with minor overhead.

- Experimental results show that WARP reduces weight writes on average to 15.25% and up to 24%, with a negligible rise in cumulative error rate (up to 0.31%). After using both WARP and TIWL, the lifetime of ODLPIM is prolonged on average to 12.59× and up to 17.73×.

II. Preliminary and Motivation

A. Online Deep Learning

Among various ODL algorithms, Hedge Backpropagation (HBP) has proved its capability of adjusting the DNN capacity dynamically to achieve fast convergence [8]. It has been applied in many fields, including click-through rate prediction and continual learning [9].

As shown in Fig. 1, a network with N layers contains three parts: the hidden layer h_i (green), the output layer o_i (red) and the classifier (yellow). The following equation demonstrates the forward propagation of ODL using HBP:

$$F(x) = \sum_{k=1}^{N} \alpha_k \cdot o_k, \forall k = 1, \cdots, N \tag{1}$$
$$o_k = softmax(h_k \cdot \Theta_k), h_k = \sigma(W_k \cdot h_{k-1}), h_0 = x$$

where W_k and Θ_k represent the weights of the hidden layer and output layer, and σ denotes the activation function. α_k is the contribution rate of the k^{th} layer, which is a learnable parameter. As shown in Eq. 2, each classifier uses the cross-entropy function to calculate the loss and then aggregate them to generate the total loss. $\hat{y}$ refers to the ground truth.

$$\mathcal{L}(F(x), \hat{y}) = \sum_{k=1}^{N} \alpha_k \cdot \mathcal{L}_{CE}(o_k, \hat{y}) \tag{2}$$

The α_k controls the training of each layer. At the beginning of training, α is uniformly distributed: $\alpha_k = 1/(N+1)$. As shown in Eq. 3, at each iteration, α_k will be decayed according to the loss value $\mathcal{L}_{CE}(o_k, \hat{y})$, which is controlled by the decay factor $\beta \in (0, 1)$. To avoid the slow learning of the deep layer due to small α, a smoothing factor $s \in (0, 1)$ is introduced: $\alpha^t = max(\alpha^t, s/N)$. Finally, the normalization of α will be performed such that the sum of α_k is 1.

$$\alpha_k^{t+1} = \alpha_k^t \cdot \beta^{\mathcal{L}_{CE}(o_k, \hat{y})} \tag{3}$$

$$\Theta_k^{t+1} = \Theta_k^t - \alpha_k \cdot \eta \nabla_{\Theta_k^t} \mathcal{L}_{CE}(o_k, \hat{y}) \tag{4}$$

$$W_k^{t+1} = W_k^t - \eta \sum_{i=k}^{N} \alpha_i \cdot \nabla_{W_i^t} \mathcal{L}_{CE}(o_i, \hat{y}) \tag{5}$$

Fig. 1(b) illustrates the backpropagation of HBP. The weight update rules of output layer and hidden layer are shown in Eq. 4 and Eq. 5, respectively. The error is backpropagated from each classifier independently. In this way, knowledge can be shared between shallow and deep layers. Therefore, enabling the network to automatically modify its effective neuron capacity in a dataflow-driven manner.

B. ReRAM-based analog computing

A ReRAM cell is a two-port element that indicates different states by its conductance, with metal electrodes at two ends and a metal-oxide layer in the middle. The resistance of a cell can be changed by applying a voltage of specific polarity, magnitude and duration onto two metal electrodes. The crossbar, as the core component of the ReRAM-based accelerator, is constructed by mutually perpendicular wordlines (WLs) and bitlines (BLs) with ReRAM cells sandwiched at each crosspoint. During the computation, the weights G are mapped to the conductance of the ReRAM cells. By applying voltage to WL, the input vector $\overrightarrow{V}$ is fed and the current is summed at each BL according to Kirchhoff's Current Law: $i_{out_k} = \sum_{i=1}^{n} V_i \cdot G_{ik}$. This analog calculation reduces the algorithmic complexity of the VMM from $O(n^2)$ to $O(1)$.

C. Motivation

1) Non-essential weight update for HBP: As shown in Fig. 2, the HBP algorithm automatically modifies the adequate neuron capacity of the network by controlling the distribution of α. However, when performing online deep learning on PIM architecture, the weights of all hidden layers are updated regardless of the value of α. Due to the low contribution rate, the inactive layers gain very little knowledge and have less impact on the final output. Therefore, we aim to reduce crossbar writes and extend the lifetime by restraining weight updates in inactive hidden layers while maintaining network performance.

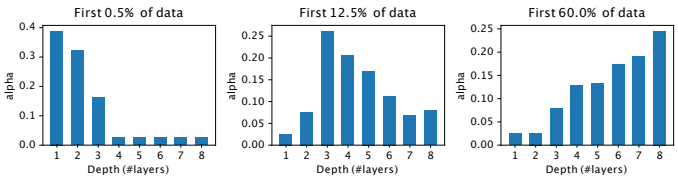


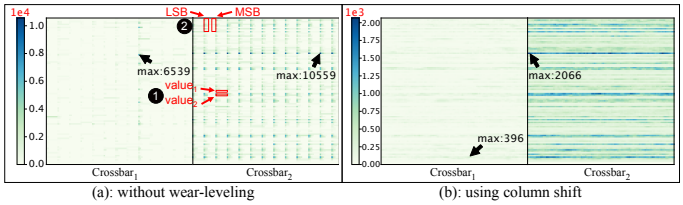
Fig. 2. Evolution of α distribution of the layers using HBP on Higgs dataset.

2) Unbalanced writes between crossbars: Table I shows the comparison of the previous wear-leveling schemes in the aspect of storage & algorithm overhead, application scalability, granularity and effectiveness. Previous studies did not realize that the quantization in the PIM system causes severely unbalanced writes. As shown in the right subplots of Fig. 3 (a), quantization drops tiny weight updates due to insufficient quantization precision, leading to unbalanced updates within the weight matrix (❶). In the right subplots of Fig. 3 (a), after quantization, the difference between the update frequency of the MSB and LSB leads to unbalanced writes in each weight data (❷). Comparing the max write number in Fig. 3 (a), we found that the write imbalance is severe. Besides, CS, RS and RCS perform wear-leveling within a crossbar, resulting in limited effectiveness. Take CS as an example, although Fig. 3(b) shows the effectiveness of CS, there is still a large difference between

TABLE I
COMPARISON OF WEAR-LEVELING SCHEMES.

	Storage Overhead	Algorithm Overhead	Application Scalability	Granularity	Effectiveness
CS [5]	Very low, require to store the column offset	None, shift directly	High, algorithm-independence	bitline group	limited, wear leveling within a crossbar
RS [6]	High, require a register in every row of matrix	High, sort by row writes	low, the size of register depends on application	row of matrix	limited, wear leveling within a matrix
RCS [7]	Very high, require the conductance bounds of cells	Very high, solve a linear optimization in every crossbar	High, algorithm-independence	subblock of a crossbar	limited, wear leveling within a crossbar
TIWL	low, require 160b for a crossbar	low, sort by crossbar writes	High, algorithm-independence	crossbar	good, an orthogonal method that achieves wear leveling between crossbar

crossbars. Thus, it is necessary to design an inter-crossbar wear-leveling scheme to improve the lifetime of the whole PIM accelerator.



(a): without wear-leveling (b): using column shift

Fig. 3. Comparison of write distribution of two crossbars in the system with/without column shift, using 8-bit quantization precision and 1-bit ReRAM.

III. RERAM-BASED ODL ACCELERATOR

A. Write-Optimized Parameter Update Scheme

We found that those layers with tiny α have small updates and low contribution to the final result, which means that the benefit of updating these non-critical layers is little compared to layers with large α. Therefore, at the algorithm level, we propose a write-optimized parameter update (WARP) scheme to reduce unnecessary weight updates in hidden layers during ODL on the PIM accelerator (the pseudo-code is shown in Algorithm 1). A layer is defined as a non-critical layer when it's α smaller than threshold ε. We introduce ε as a hyperparameter to adjust the reduction of weight updates. Generally, we set it to $1/N$. If it is set too large, the model convergence will be affected. We also use an update interval γ (generally set to 2) to avoid the network failing to converge. The output layer is the "judge" of the hidden layer, so we cannot restrict its weight updates (Θ). Otherwise, it will exacerbate the performance.

Algorithm 1: Write-Optimized Parameter Update Scheme for Online Deep Learning

Input: lr: η, decay factor: β, smoothing factor: s, number of layers: N, threshold ε, interval: γ
Output: DNN model $F(X)$

Initialize: $F(X) = DNN$ with N layers;
$\alpha_k = \frac{1}{N+1}, \forall k = 1, \cdots, N$;
foreach *data instance* $X_i \in$ *dataset* **do**
  predict $y_i = F(X_i) = \sum_{k=1}^{N} \alpha_k \cdot o_k$ as per **Eq. 1**;
  for $k \leftarrow 1$ **to** N **do**
    calculate loss with label $\hat{y_i}$: $\mathcal{L}_{CE}(o_k(X_i), \hat{y_i})$;
  end
  for $k \leftarrow 1$ **to** N **do**
    gradient backpropagation of classifier o_k ;
    update $\Theta_k, \forall k = 1, \cdots, N$ as per **Eq. 4** ;
    if $\alpha_k \geq \varepsilon$ ***or*** $i \bmod \gamma == 0$ **then**
      update W_k as per **Eq. 5** ;
    end
  end
  update, smoothing and normalize α_k;
end

B. Table-Based Inter-Crossbar Wear Leveling Scheme

1) Design details: Previous works focus on intra-crossbar wear-leveling which are difficult to address the severely unbalanced writes between crossbars. So, we propose a table-based inter-crossbar wear-leveling (TIWL) scheme at the hardware level to achieve wear-leveling by remapping "hot" logical arrays to less worn physical arrays. There are two metadata related to TIWL for each crossbar, **interval wear count (IWC, 32-bit)** and **total wear count (TWC, 64-bit)**, indicating the hotness and the wearing degree of the crossbar. To reduce the storage overhead, we use the summation of cell writes during an interval (IWC) to indicate the hotness of a crossbar. IWC is stored in a register of the crossbar as the temporality and frequent updates of it. Meanwhile, TWC is important metadata that records the total number of cell writes during the crossbar's lifetime. Thus, it is stored concentratively in a dedicated crossbar group safely.

In the following discussion, we define two array IDs. **Logical Array ID (LAID)** is a virtual reference assigned by the system to access the physical array. **Physical Array ID (PAID)** identifies a physical crossbar in the PIM system. TIWL consists of two phases: running and remapping & data swapping, As shown in Fig. 4. In the running phase, the write driver accumulates the total number of cell writes to the IWC register at each write operation. Compared with previous works, TIWL does not need to record the aging status of each WL or BL, nor get the temperature of cells. Thus, the storage and computational overhead are reduced. In the remapping & data swapping phase, the IWCs of all crossbars are accumulated into corresponding TWCs. Then TIWL rebuilds the mapping table and performs data swapping. Data swapping is hidden in the weight update, which reduces the write amplification by overlapping the writes of data swapping and weights update.

As shown in Table I, unlike RS limited in a specific domain,

TIWL can achieve system-level wear-leveling for any PIM application. Compared to RS and RCS, TIWL needn't record the write counts of each WL or BL, nor get the temperature of cells, resulting in the low overhead of storage and computational. By customizing the granularity of mapping such as a processing element (multiple crossbars), TIWL can achieve a balance between mapping overhead and mapping profits which is architecture-friendly. In this paper, the granularity is a crossbar. It is important that TIWL is an orthogonal scheme that can be used together with other intra-crossbar wear-leveling schemes, such as CS, RCS.

2) Storage overhead analysis: Suppose building a ReRAM-based PIM system with a capacity of 4 GB using the crossbar of size 128×128, the storage overhead is acceptable, 24 MB (0.59% of 4 GB) for wear metadata and 16 MB (0.15% of 4 GB) for the mapping table. When the crossbar size is increased to 512, the storage overhead for wear metadata and mapping tables drops to 1.14 MB (0.03% of 4 GB) and 1.5 MB (0.04% of 4 GB). The metadata of a crossbar is fixed to 96 bits which is much smaller than RS ($N \times T$, where N denotes row number and T is bits width of register).

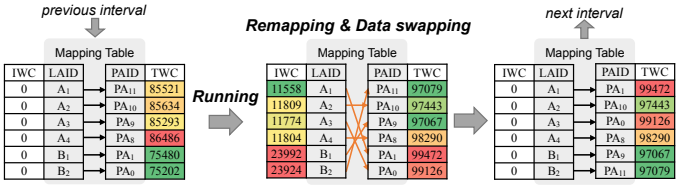


Fig. 4. The basic workflow of table-based inter-crossbar wear leveling scheme and the structure of the mapping table. The red indicates heavy updates, while the green is the opposite.

C. Architecture Design

We first present an overview of the ODLPIM architecture (shown in Fig. 5). The ACU (❸) is used to process the results of crossbars from different tiles, including ReLU activation, pooling, accumulation, and vector operation for back-propagating. Torus routing (❶) [10] is used for interconnection

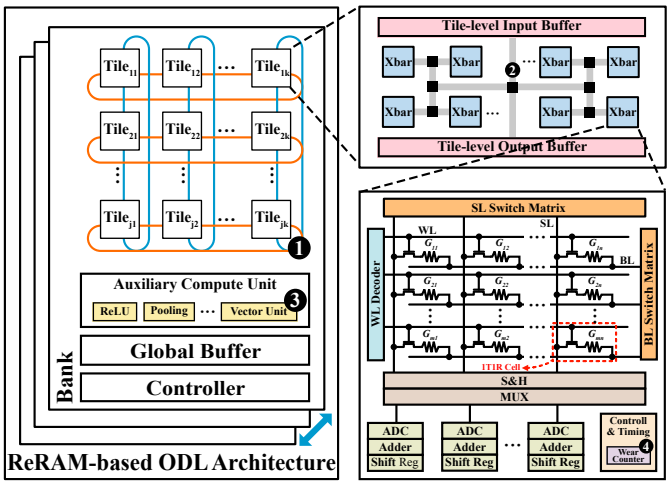


Fig. 5. The ODLPIM architecture overview.

TABLE II
DATASET DESCRIPTION.

Dataset	#Instances	#Features	#Classes	Type
Higgs	5M	28	2	Stationary
Susy	5M	18	2	Stationary
Inf-MNIST	3M	784	10	Stationary
CD	4.5M	50	2	Concept Drift

between tiles. Each Tile is composed of multiple crossbars and tile-level input/output buffer, interconnected by H-Tree (❷). To mitigate write interference and sneak currents [11], 1T1R cells are used to construct crossbars. Within the crossbar controller, the IWC is stored in the wear counter (❹).

ODLPIM adopts a two-level (Tile/Crossbar) hierarchy design, instead of the popular three-level (Tile/PE/Crossbar) PIM architecture [2], [12], [13]. As analyzed in Section III-B1, the crossbars belonging to a logical array may be distributed in multiple tiles after remapping. To reduce unnecessary communication overhead, we remove the PE hierarchy and use Torus routing for interconnection between tiles. ODL has a smaller bandwidth requirement (commonly $batchsize = 1$) than offline batch training. Moreover, the decentralized computation can mitigate bandwidth contention within the tile.

The computation of gradient ∇W requires the intermediate data x as input: $\nabla W_l = x_{l-1}(\delta_l)^\top$, so x should be stored in the forward stage. There are two drawbacks if we use the crossbar to generate ∇W [1] in the ODL situation: (1) Rewriting the crossbar frequently introduces energy overhead and reduces the lifetime. (2) Storing the striped intermediate data in the crossbar would waste space and aggravate the wear unbalance. In ODL, intermediate data storage space is smaller than in OBL. For training of 16-layers network (of 100 widths each) using HBP, 9KB is sufficient. Therefore, we store x in the global buffer and use the ACU to generate ∇W, similar to [2].

IV. EVALUATION

A. Experiment Setup

1) Benchmarks: Common datasets (cifar10, cifar100, etc.) are too small to simulate large-scale sequential data streams, we used million-scale datasets from [8] which is widely used in ODL situations to evaluate network performance. Specifically, we evaluated the sensitivity of the model to concept drift in the artificial dataset CD, which contains three concept patterns, P_1, P_2, and P_3. Each concept pattern accounts for 1/3 of the overall dataset. Table II shows the details of the dataset.

2) Simulation Parameters: We developed an in-house simulator by utilizing the LibTorch interface to evaluate the performance of ODLPIM. Particularly, system-level information is recorded to evaluate TIWL scheduling of the crossbars. The area and power metrics are retrieved from [12], [14] in 32 nm. Table III shows the energy and area of a tile. We use our simulator to evaluate an 8-layer DNN training (each layer has 100 units) in an online setting, with 8-bit weight quantization precision and 12-bit array input bit width. As a comparison, the software baseline is trained using PyTorch. For model hyperparameters, learning rate $\eta = 0.01$, decay factor $\beta = 0.99$, and smoothing factor $s = 0.2$. The threshold ε

TABLE III
ODLPIM TILE HARDWARE SPECIFICATION

Component	Parameter	Spec	Power (mW)	Area (mm^2)
ADC	resolution frequency number	8-bit 1.2 GHz 32	64	0.0384
DAC	resolution number	1-bit 8×128	4	0.00017
S & H	number	8×128	0.01	0.00004
crossbar array	number size bit/cell	8 128×128 1	2.43	0.00069
buffer	capacity	2Kb	0.00097	0.00326

TABLE IV
CERS OF DNNS AND WRITE REDUCTION ON DIFFERENT DATASETS

	Cumulative Error Rate (CER)			Write Reduction of WARP
	software baseline	w/o WARP (vs. baseline)	w/ WARP (vs. w/o WARP)	
Higgs	0.2632	0.2625 (-0.0007)	0.2687 (+0.0062)	21%
Susy	0.2003	0.2004 (+0.0001)	0.2008 (+0.0004)	7%
Inf-MNIST	0.0186	0.0710 (+0.0524)	0.075 (+0.004)	24%
CD	0.4122	0.4128 (+0.0006)	0.4146 (+0.0018)	9%

for WARP is determined by the number of network layers: $\varepsilon = 1/N$. Wear-leveling is performed every 1000 iterations.

B. Performance & Write Reduction Results

Due to the variability of streaming data, a static validation set cannot evaluate the performance of dynamic models. Thus, we demonstrate the performance of the model by evaluating the cumulative error rate (CER). The CER is given by: $CER = \frac{num\ of\ incorrect\ predictions}{total\ num\ of\ predictions}$. As shown in Table IV, compared to the software baseline, the CERs for ODL without WARP has a slight increase on average of 1.31% due to quantization. Although WARP reduces the non-critical weight updates, it achieves a resemble performance and convergence as without WARP, with an acceptable rise in CER up to 0.4%. Besides, we trace all crossbar cells' writes during training through our simulator and find that WARP achieves a write reduction by 15.25% on average compared to that without using WARP, as shown in Table IV.

C. Wear Leveling Analysis

In the follow-up discussion, we focus on the effectiveness of WARP and TIWL in ODLPIM, so we set the case without using WARP and TIWL as the comparison (PIM baseline) and set the number of data instances to 0.5M which is enough for evaluation. **We define the $MaxWrt$ of the crossbar as the number of writes of the worse cell in it.** Fig. 7 shows the variation of $MaxWrt$ distribution for all crossbars during ODL. There are nine lines, from top to bottom, indicating the standard deviation boundaries on a normal distribution: [maximum, $\mu + 1.5\sigma$, $\mu + \sigma$, $\mu + 0.5\sigma$, μ, $\mu - 0.5\sigma$, $\mu - \sigma$, $\mu - 1.5\sigma$, minimum], and the percentile of data distribution (defined as R) corresponding to each line is: [100%, 93%, 84%, 69%, 50%, 31%, 16%, 7%, 0%]. We define the difference between the maximum and minimum $MaxWrt$ as $\Delta MaxWrt$ to evaluate the write imbalance of ODLPIM. To predict how many data instances the ODLPIM can support during the whole lifetime, we extend these lines using linear fitting (red part) to the endurance of a ReRAM cell (10^8 according to [5]).

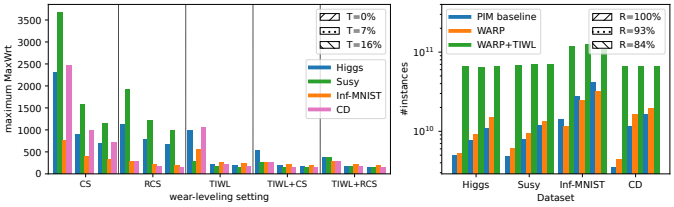


Fig. 6. (a) The maximum MaxWrt in the system with different wear-leveling settings and tolerances T. (0.5M instances). (b) Lifetime extension of WARP and TIWL with different R.

Although most logical arrays have different update hotspots, TIWL makes these hotspots evenly dispersed to different crossbars, ultimately achieving inter-crossbar wear-leveling for the entire PIM system. Fig. 7 demonstrates that TIWL significantly reduces the $\Delta MaxWrt$. Specifically, the reduction of $\Delta MaxWrt$ in four datasets averages about 6.8×10^4, up to 1.2×10^5. The violin plot in Fig. 8 directly illustrates the probability density of $MaxWrt$ when the training is over. Since WARP reduces the non-essential weight updates, it slows down the growth of MaxWrt. The mean values of $MaxWrt$ are reduced by 13.78% (Higgs), 12.33% (Susy), 2.57% (Inf-MNIST), and 12.9% (CD). A slight increase of maximum $MaxWrt$ is shown in Inf-MNIST dataset after using WARP which may be caused by the volatility of weight update, but the distribution of $MaxWrt$ concentrate on a smaller value compared to baseline. Fortunately, after combining WARP and TIWL, the mean values of $MaxWrt$ reduce by 75.86% (Higgs), 74.23% (Susy), 51.05% (Inf-MNIST), and 67.91% (CD), even lower than the 25^{th} percentile in some cases.

We also conducted comparison experiments with CS to demonstrate TIWL's excellent wear-leveling power. The tolerance T of a crossbar is defined as the percentage of worst cells that can be accepted. We collect the minimum $MaxWrt$ amount of the T% worst cells for each crossbar, then choose the maximum amount for them to indicate the maximum $MaxWrt$ of the worse crossbar in the system. As shown in Fig. 6(a), the effect of CS is the poorest because it only performs wear-leveling inside the crossbar. RCS achieves balanced writes among different rows, so it is more effective than CS. What's more, TIWL achieves wear-leveling between crossbars, it decreases the maximum $MaxWrt$ on average of 58.35% (T=0%), 68.36% (T=7%) and 66.07% (T=16%), compared to CS. As TIWL is an orthogonal method that can reinforce the wear-leveling performance of CS/RCS. For CS/RCS, the average reduction of maximum $MaxWrt$ is 81.36%/37% (T=0%), 74.26%/41.01% (T=7%) and 71.38%/40.76% (T=16%).

D. Lifetime Extension

We assume that a crossbar breaks down when it's $MaxWrt$ reached the endurance. The maximum number of training

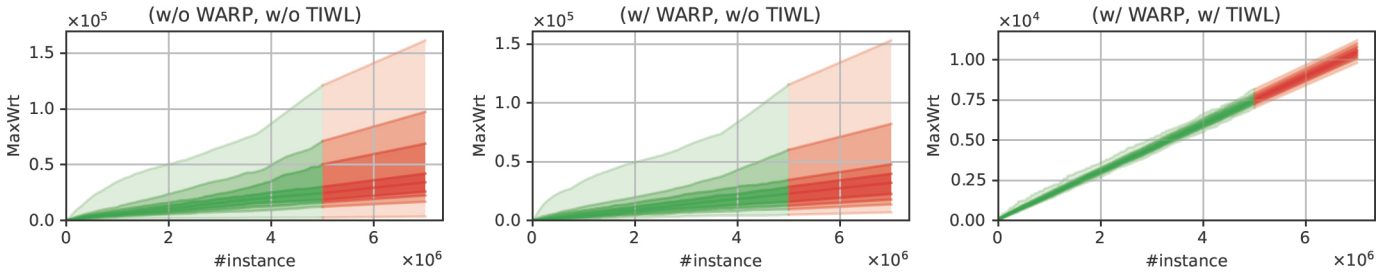


Fig. 7. Variation in the distribution of MaxWrt of the crossbar, where the green part is the real data and the red part is the predicted trend using linear fitting. To reduce the space, only the results of the Higgs dataset are shown and others have similar characteristics.

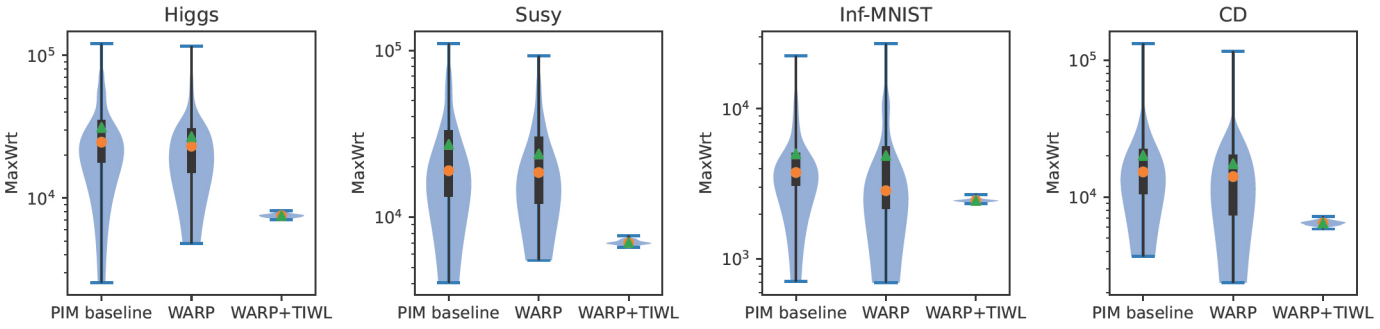


Fig. 8. The MaxWrt distribution of crossbar when training finished, where the green triangle indicates the average and the orange circle indicates the median.

instances is chosen as the evaluation metric while R% of the crossbars working properly. We assume that the PIM system cannot guarantee the model accuracy when R<84%, so we focus on the top three lines in Fig. 7 (R=100%, R=93% and R=84% respectively) and use linear fitting to predict the maximum number of training instances. Fig. 6(b) demonstrate the lifetime extension of WARP and TIWL. WARP extends the lifetime on average by 9.65% (R=100%), 16.75% (R=93%) and 11.56% (R=84%). However, in the Inf-MNIST dataset, WARP does not show the expected lifetime extension due to the volatility of weight update, which can be improved by selecting the optimal hyperparameter of WARP. Thanks to inter-crossbar wear-leveling, WARP+TIWL achieves 3.76× (R=100%), 5.85× (R=93%) and 12.59× (R=84%) lifetime extension on average.

V. Conclusion

To the best of our knowledge, we have been the first to present an efficient online deep learning PIM accelerator, called ODLPIM, with an algorithm-hardware co-optimization to prolong its lifetime. We use WARP to reduce unnecessary weight updates and TIWL for inter-crossbar wear-leveling. The experimental results show that WARP achieves an average of 15.25% write reduction. For lifetime extension, compared to PIM baseline, WARP extends on average 9.65% (R=100%), 16.75% (R=93%) and 11.56% (R=84%), while WARP+TIWL extends 12.59×, 5.85× and 3.76×, respectively.

Acknowledgment

This work was sponsored in part by the National Natural Science Foundation of China under Grant 61832007, Grant 61821003 and Grant 62172178, in part by the pre-research project No.31511090201.

References

[1] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017, pp. 541–552.

[2] K. Qiu, Z. Zhu, Y. Cai, H. Sun, Y. Wang, and H. Yang, "Mnsim-time: Performance modeling framework for training-in-memory architectures," in *AICAS*, 2021, pp. 1–4.

[3] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+neurosim v2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE TCAD*, vol. 40, no. 11, pp. 2306–2319, 2021.

[4] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys*, vol. 46, no. 4, pp. 1–37, 2014.

[5] Y. Cai, Y. Lin, L. Xia, X. Chen, S. Han, Y. Wang, and H. Yang, "Long live time: improving lifetime for training-in-memory engines by structured gradient sparsification," in *DAC*, 2018, pp. 1–6.

[6] W. Wen, Y. Zhang, and J. Yang, "Renew: Enhancing lifetime for reram crossbar based neural network accelerators," in *ICCD*, 2019, pp. 487–496.

[7] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *AICAS*, 2020, pp. 11–15.

[8] D. Sahoo, Q. Pham, J. Lu, and S. C. H. Hoi, "Online deep learning: Learning deep neural networks on the fly," in *IJCAI*, 2018, p. 2660–2666.

[9] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.

[10] H. Matsutani, M. Koibuchi, and H. Amano, "Performance, cost, and energy evaluation of fat h-tree: A cost-efficient tree-based on-chip network," in *IPDPS*, 2007, pp. 1–10.

[11] B. Wu, D. Feng, W. Tong, J. Liu, S. Li, M. Yang, C. Wang, and Y. Zhang, "Aliens: A novel hybrid architecture for resistive random-access memory," in *ICCAD*, 2018, pp. 1–8.

[12] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016, pp. 14–26.

[13] G. Yuan, P. Behnam, Z. Li, A. Shafiee, S. Lin, X. Ma, H. Liu, X. Qian, M. N. Bojnordi, Y. Wang, and C. Ding, "Forms: Fine-grained polarized reram-based in-situ computation for mixed-signal dnn accelerator," in *ISCA*, 2021, pp. 265–278.

[14] H. Guo, L. Peng, J. Zhang, Q. Chen, and T. D. LeCompte, "Att: A fault-tolerant reram accelerator for attention-based neural networks," in *ICCD*, 2020, pp. 213–221.